Advantage™ EDBC® for CA-IDMS®

Installation and Operations Guide

Computer Associates™

Contents

Chapter 4: Preparing for Installation

Chapter 5: Installing the Gateway

Chapter 6: Configuring the EDBC Server

Chapter 9: Using Database Procedures

Chapter 1

Introduction

EDBC for CA-IDMS (Integrated Database Management System) allows EDBC users to access data stored in CA-IDMS databases. It also allows EDBC users on a client to develop database applications on a UNIX or Microsoft Windows platform that can be executed against CA-IDMS data without requiring additional programming.

In This Guide

In this guide the term:

- *CA-IDMS gateway* or *gateway* is synonymous with *EDBC for CA-IDMS*
- *OpenSQL* is synonymous with *EDBC OpenSQL*
- *Star* is synonymous with *Ingres/Star*

The *EDBC for CA-IDMS Installation and Operations Guide* serves as both a learning tool and a permanent reference guide.

This guide explains how to perform the following tasks:

- Prepare for the installation
- Install EDBC and the gateway to CA-IDMS
- Verify that EDBC and the gateway have been successfully installed
- Operate and maintain EDBC and the gateway

This guide provides an overview of the CA-IDMS gateway and the other components with which the gateway interacts. It also covers the following topics:

- EDBC function and architecture
- Configuring EDBC
- Using CA-IDMS database procedures
- Using CA-IDMS table procedures

The appendixes include information on the following topics:

- EDBC server and gateway logical symbols and IIPARM Clist
- Customization
- Coding database procedures
- Multiple Central Version support

Audience

The first six chapters of this guide are addressed to the individual who is responsible for installing the CA-IDMS gateway and for supporting EDBC users who will access CA-IDMS data through the gateway. It assumes a working knowledge of OS/390, TSO/E, ISPF, and JCL, and familiarity with CA-IDMS.

The "Introduction" chapter and the "Gateway Architecture and Operation" and "Configuring and Starting Net" chapters are directed to the end user who will be using remote applications to access CA-IDMS data through the gateway.

This guide may also be useful to individuals who are already familiar with CA-IDMS and OS/390 and who want to understand how the CA-IDMS gateway works in this environment.

Conventions

This guide employs several conventions, described in this section, to make identifying information easier.

Terminology

This guide observes the following distinction in terminology:

- A *command* is an operation that you execute at the operating system level. An extended operation invoked by a command is often referred to as a *utility*.
- A *statement* is an operation that you embed within a program or database procedure, or execute interactively (for example, using the Terminal Monitor or the SQL Test window in Visual DBA).

  A statement can be written in Ingres/4GL, a host programming language (such as C), or a database query language (such as SQL or QUEL).

Query Languages

The industry standard query language, SQL ISO Entry SQL92, is used as the standard query language throughout this guide. For details about the settings required to operate in compliance with ISO Entry SQL92, see the online *SQL Reference Guide*.

Syntax and User Input

When representing syntax and user input, the following conventions are used:

Convention	Usage
Boldface	Indicates any text that you must type as shown.
Italics	Indicates a variable name or placeholder for which you must supply an actual value—this convention is used in explanatory text, as well as syntax.
Case Sensitivity	System command and environment variable names may be case-sensitive, depending on the requirements of your operating system.
[ ] (square brackets)	Used to enclose an optional item.
{ } (curly braces)	Used to enclose an optional item that you can repeat as many times as appropriate.
| (vertical bar)	Used between items in a list to indicate that you should choose one of the items.

Example

The following example illustrates some of these conventions:

cd *directory*

The **cd** portion is in bold, so you enter it exactly as shown. The *directory* portion is in italic, indicating a placeholder that you must replace with a value that is applicable to your system.

Chapter

2

Overview of EDBC for CA-IDMS

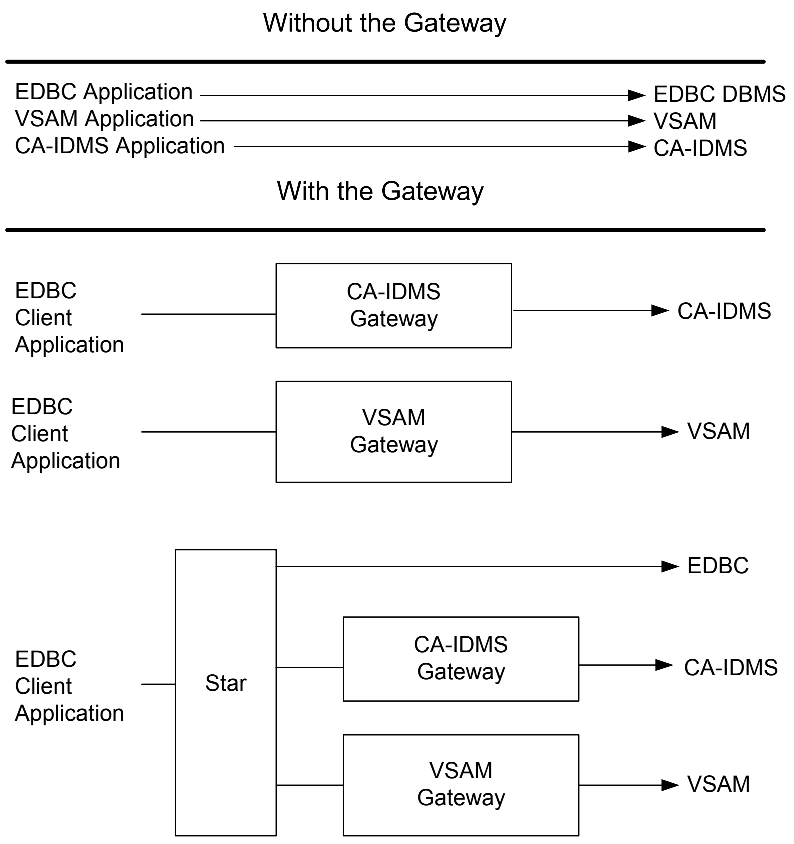
EDBC for CA-IDMS is part of a family of products that provides access to data across a wide range of hardware platforms, operating systems, and database management systems.

This chapter provides an overview of the gateway and the other components with which the gateway interacts. The "Gateway Function and Architecture" chapter discusses specific aspects of the CA-IDMS gateway.

The EDBC Solution

EDBC for CA-IDMS gateway is one of a family of gateway products that provide a consistent way to access heterogeneous databases. The gateways permit EDBC users to run applications against local data and against data that resides in foreign relational and non-relational database management systems. *Relational gateways* provide bridges to relational database management systems such as CA-Datacom, IBM DB2, Rdb, and ALLBase SQL. *Non-relational gateways* provide bridges to non-relational database management systems such as IMS and VSAM.

The following figure shows an example of how a gateway provides these bridges:



A Gateway Installation

A gateway installation, running under OS/390 at a CA-IDMS site, includes the following components:

- Gateway to access CA-IDMS tables
- EDBC server to permit access between OS/390 and user interfaces residing on a different computer system
- Star to allow an EDBC client to access multiple databases simultaneously
- Several client user interfaces, such as the Terminal Monitor, ODBC, ADO, and OLE/DB

All components communicate through the Global Communications Architecture (GCA), which resides at each client and server location.

The following sections describe these components in detail.

The CA-IDMS Gateway

CA-IDMS files are major repositories of corporate data. The gateway gives EDBC clients transparent access to CA-IDMS data. It also enables programmers to use local client tools to develop and deploy applications that process CA-IDMS data.

Combined with other components, the gateway:

- Provides transparent access to CA-IDMS data

  The gateway allows you to run client user interfaces to access tables and views stored in a CA-IDMS file. This access is transparent to the end user. The CA-IDMS data appears to an EDBC client as though it is stored in a local database.

- Runs as a standard CA-IDMS application

  The gateway runs as a standard CA-IDMS application by translating queries from EDBC client user interfaces into native CA-IDMS requests. CA-IDMS executes the queries and returns the results back to the gateway. The gateway collects and packages the responses into syntax meaningful to the client user interfaces.

- Enhances programmer productivity

  The gateway allows a client to build portable database applications for CA-IDMS using client-based application development tools, such as Power Builder or Visual Basic. These applications can run unmodified against any server or gateway, and can coexist with existing CA-IDMS applications that access and update this data.

- Unifies data access with a single language

  The gateways allow users to work with data using ADO. This allows them to access CA-IDMS, Ingres, or any other DBMS using the same language.

- Integrates desktops, workstations, and foreign mainframes into the OS/390 environment

  The gateway allows EDBC clients to retrieve and process foreign data, and develop database applications while working on computers such as Sun, HP, and PCs running under operating systems such as UNIX and Microsoft Windows.

- Supports distributed processing

  With the addition of Star, the gateway allows users to join data from diverse databases, such as CA-IDMS, CA-Datacom, VSAM, and IBM DB2, and use the resulting tables as if they were derived from a single source.

EDBC Server

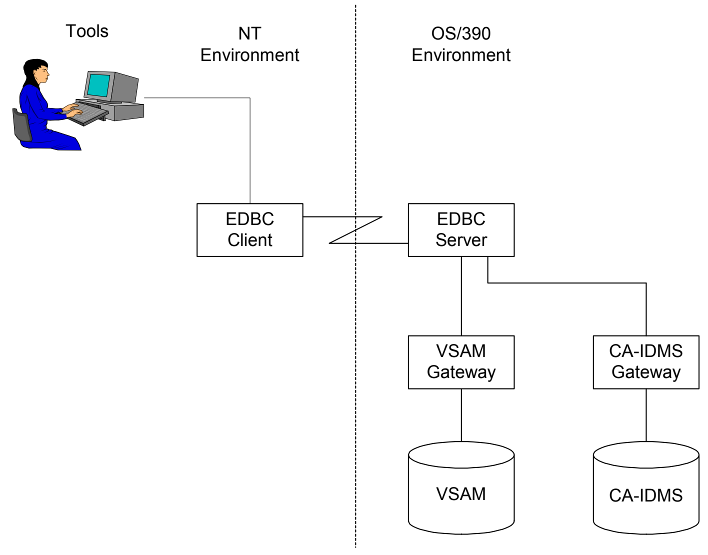
EDBC allows you to access data in two modes:

- In *local* mode, the EDBC user interfaces and the target database reside on the same computer system.
- In *remote* mode, the EDBC user interfaces and the target database reside on different computer systems. These systems must be linked with a computer network. The system where the user interfaces reside is called the *client*. The system where the target database, and, if necessary, the gateway, resides is called the *server*.

EDBC is a network application that enables EDBC users to access databases on remote platforms. With the gateways, EDBC permits communication between components running on many types of machines, under diverse operating systems, including OS/390, UNIX, and Microsoft Windows.

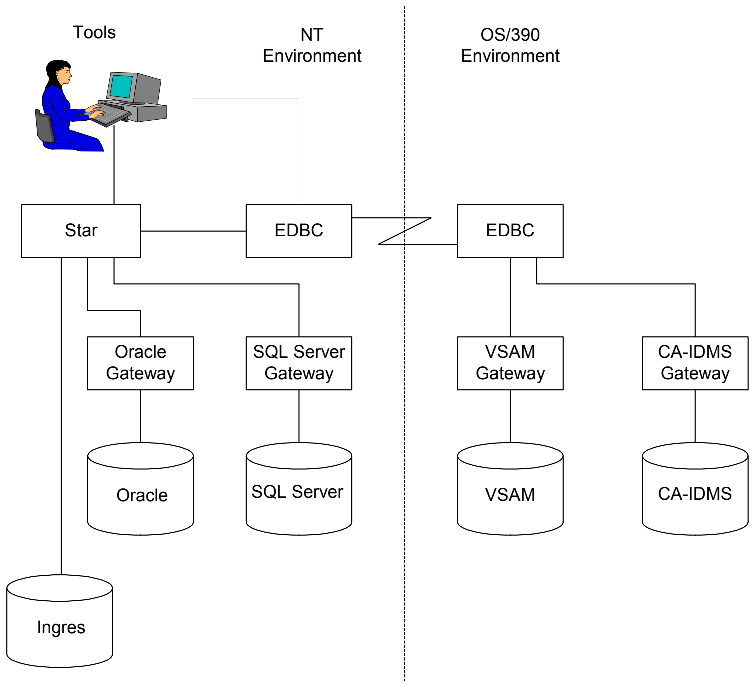
The EDBC software is installed on each machine that represents a node on the network. When the user invokes an EDBC tool or application on his local node, EDBC passes the query to the database on the proper remote node, using the appropriate protocol. EDBC supports multiple protocols, including SNA LU0, SNA LU62, TCP/IP, SNS/TCP, and CCI. CCI is supported only on OS/390.

The following illustration shows a configuration connecting an EDBC client on an NT system with an OS/390 system:



Ingres/Star

Star is a distributed information manager that allows an EDBC user to access multiple databases simultaneously. Star makes it possible to create a distributed database containing data from EDBC databases and repositories such as, CA-Datacom, RMS, VSAM, and IBM DB2. From a user's perspective, a distributed database (DDB) has all the characteristics of a local relational database. The base tables, however, remain physically located in multiple databases that can be of one or more types, residing on one or more systems. The following illustration shows a configuration similar to the one in the previous illustration, but with the addition of Star. See the *Star User Guide* for additional information about Star and distributed databases.



Security System Support

The gateway fully supports the following major OS/390 security systems:

- IBM Resource Access Control Facility (RACF)
- Computer Associates Access Control Facility 2 (CA-ACF2)
- Computer Associates Top Secret Security (CA-TSS)

For more information about these systems, see the "Configuring and Starting Net" chapter.

User Interfaces

Any application development interface that is ADO, OLE/DB, ODBC, or JDBC compliant can be used to access CA-IDMS data through the gateway.

The gateway supports access from EDBC user interfaces to CA-IDMS tables. However, the converse is not true. A user running a CA-IDMS application cannot use the gateway to access data stored on an EDBC client.

Structured Query Language (SQL)

EDBC operates on data using the Structured Query Language (SQL). SQL consists of high-level descriptions of actions to be performed against data, such as select and prepare. SQL requires no instructions for accessing data; it navigates its own way through the database.

The gateways use OpenSQL, which is a subset of Ingres/SQL. OpenSQL is highly portable, because it is designed to open applications to many different types of databases. All EDBC products support OpenSQL. Applications written in SQL must be rewritten in OpenSQL for use with the gateways. OpenSQL is documented in the *EDBC OpenSQLReference Guide*.

For more information about OpenSQL and the gateway, see the "Working with CA-IDMS Data" chapter.

Chapter

3

Gateway Function and Architecture

This chapter provides a technical overview of the gateway, EDBC server, and the gateway user interfaces that run on OS/390. The chapter describes how the CA-IDMS gateway is structured, and how it interacts with the other components.

Gateway Functions and Structure

The gateway is implemented as a standard CA-IDMS application that runs under OS/390. It communicates with a CA-IDMS Central Version (CV) and does not affect the internal operations of CA-IDMS. The CA-IDMS Central Version continues to perform all database processing, and to manage and access its own tables, views, indexes, and system catalogs. From the perspective of the CA-IDMS system, queries from the gateway are identical to queries from any standard CA-IDMS application.

The gateway creates and maintains its own system catalogs that are required to support EDBC interfaces. These system catalogs consist primarily of views that are drawn from the CA-IDMS system tables and do not interfere with them in any way. The operation of the gateway is transparent to CA-IDMS.

Gateway Functions

With proper authorizations within CA-IDMS, you can create, query, update, or report data in a CA-IDMS Central Version using EDBC user interfaces. When you make a database request, the gateway:

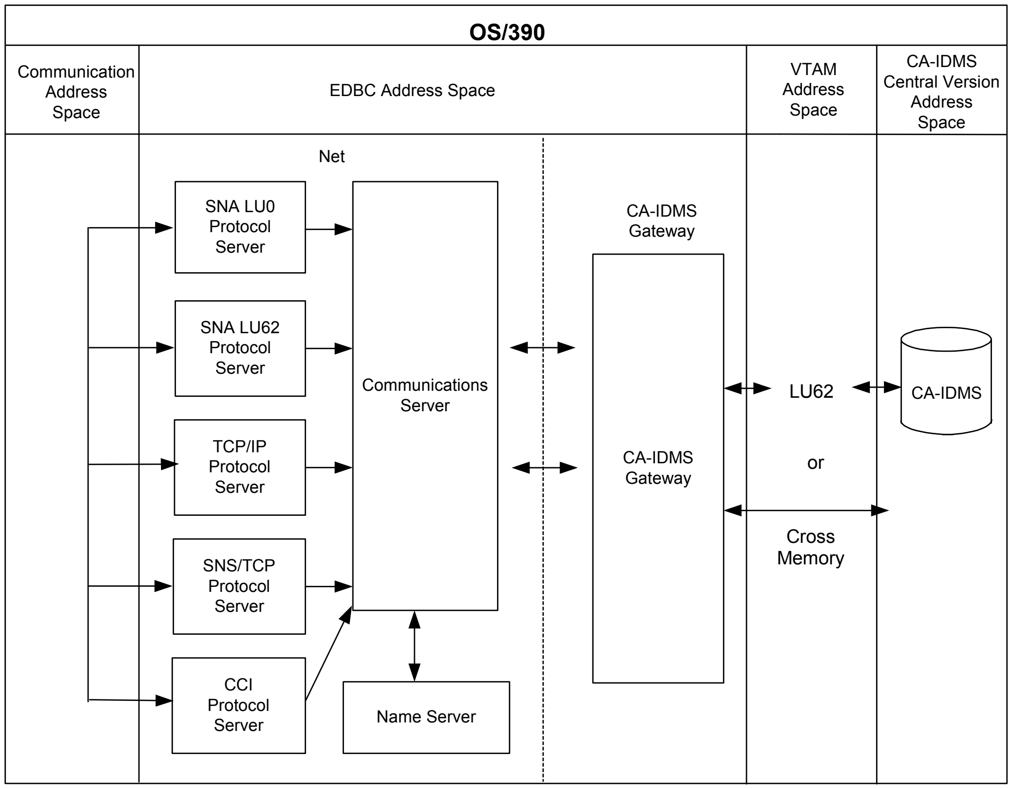
- Connects to the specified CA-IDMS Central Version
- Translates OpenSQL to the CA-IDMS version of SQL, converting EDBC data types into CA-IDMS data types as necessary
- Passes the direct execute immediate statement directly to CA-IDMS without modification
- Prepares, describes, and opens any query requiring data
- Translates into a cursor any select statement that is not already expressed as a cursor

- Translates the response from CA-IDMS SQL and converts CA-IDMS data types into EDBC data types as necessary
- Converts any error messages into generic error messages

EDBC Architecture

EDBC employs a multi-user, single address space architecture. This takes advantage of the OS/390 multi-tasking operating system.

The gateway, protocol servers, name server, and communication server operate in a single address space in OS/390 and are referred to collectively as the EDBC server. VTAM, TSO, and one or more CA-IDMS Central Versions are installed in other, separate address spaces, as shown in the following illustration. It depicts the flow of information when a remote user accesses the gateway. The illustration in the Local Access section shows the flow of information when the gateway is accessed by a local user:



Remote Access

This section describes the use of OS/390 address spaces when a remote user connects to the gateway.

Communication Address Space

When a connection request arrives from a remote EDBC client, it goes first to a communication address space. This communication address space typically implements the lower three layers (Network, Data Link, and Physical) of the Open Systems Intercommunication (OSI) standard.

Different supported protocols require the appropriate communication address space. The following communication address spaces are used:

- IBM VTAM address space that supports the SNA LU0 and SNA LU6.2 protocols
- CCI address space that supports the CCI protocol
- IBM TCP/IP address space that supports the IBM TCP/IP protocol
- Spartacus KNET address space that supports the KNET TCP/IP protocol
- SNS/TCP address space that supports the TCP/IP protocol

These communication address spaces are installed and configured prior to the installation of the gateway.

EDBC Address Space

The EDBC address space includes the following components: the protocol servers, the communications server, and the name server which, together, make up the EDBC server. In addition, the EDBC address space also includes the gateway.

- Protocol servers

  The protocol servers monitor traffic over the network. These servers recognize and process incoming communication requests in their native protocols, CCI, SNS/TCP, SNA LU0, IBM TCP/IP, SNA LU6.2, and KNET TCP/IP. The protocol servers provide a multi-threaded interface to the underlying physical transport. They communicate with EDBC through in-memory message queues.

- A multi-threaded communications server

  A multi-threaded communications server (Comm server) establishes connections to the name server and to the gateway. A new thread is created for each active connection, whether incoming or outgoing. In addition, one Comm server thread listens for new communications requests. The Comm server provides communication protocol stack functions, and transport functions. It also performs cross-platform data format conversions, and manages session context. The Comm server thread services a connection to a gateway thread.

- A name server

  A single name server queues incoming connection requests and handles them serially. The name server maintains a list of the classes of communication servers (the CA-IDMS, and perhaps, the IMS Comm servers) that are installed in the gateway address space. The name server also tracks the status (active or inactive) of each Comm server in the address space.

- Gateway threads

  A gateway thread is associated with a Comm server thread. The gateway receives messages from the Comm server, translates these messages and data types, and then forwards them to the CA-IDMS Central Version. The gateway is multi-threaded so it can support multiple connections concurrently.

  The code for both the Comm server and the gateway consists of executable load modules. When you initialize the gateway address space, OS/390 loads a single copy of the Comm server and the gateway code. This code is reentrant, so multiple tasks can share a single copy of it.

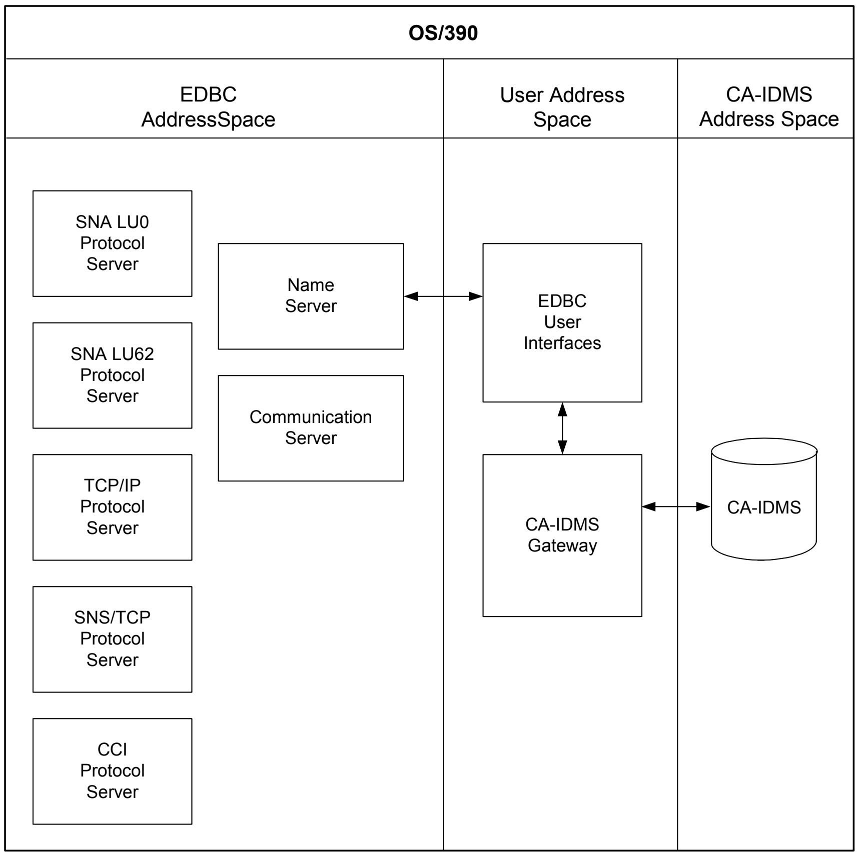
CA-IDMS Address Space

The method by which the CA-IDMS gateway interfaces to a CA-IDMS Central Version is a function of how the gateway is accessed. When the access is from a remote client, the gateway can be configured to use either a cross memory interface or an IBM VTAM LU62 line driver interface. With the cross memory option, access to multiple central versions (maximum of 50) is allowed. When accessed locally, the CA-IDMS batch CV facility is used to communicate to the the Central Version.

The CA-IDMS system handles queries from the gateway just as it would any other standard CA-IDMS application. CA-IDMS processes these statements or commands, and the results are then returned to the gateway. The gateway processes these as necessary, and returns the resulting information to the EDBC server for transmission back across the network.

Local Access

This section describes the use of OS/390 address spaces when a local user connects to the gateway. In this mode, the gateway communicates to the CA-IDMS Central Version (CV) using batch CV facilities. The following figure shows the flow of information when a local user accesses the gateway:



User Address Space

There are three EDBC user interfaces that run under OS/390:

- Terminal Monitor
- Embedded OpenSQL Preprocessor for C
- Embedded OpenSQL Preprocessor for PL/I

In the OS/390 environment, these all run in a user address space (batch or TSO). When the gateway is accessed by a user on the same OS/390 system, the gateway also runs in that user's address space. The EDBC address space is only used to establish the connection initially.

When a user issues a connect statement for a CA-IDMS Central Version, the connection request is forwarded from the EDBC user interface to the name server, which runs in the EDBC address space, as described in the EDBC address space section.

When the name server supplies the appropriate connection information, the user interface on OS/390 connects with the gateway. When accessed locally, the gateway runs in the user's address space. The gateway connects to the appropriate CA-IDMS Central Version through the batch Central Version Facility.

EDBC Address Space

EDBC must be up and running for local access because the name server runs in this address space, and it is needed to establish the connection to the gateway.

The name server is a single-threaded server that is only used at connect time. For local clients, the name server translates CA-IDMS Central Version requests into the information needed to connect to the gateway.

The Stand-Alone Back-End (SABE) parameter may be used to bypass the name server by using predefined, default server class information. The SABE parameter eliminates the requirement that the EDBC server must be running in order to provide local access. See the Verifying the Installation Functionality section in the "Installing the Gateway" chapter for information on how to use this parameter.

CA-IDMS Address Space

The interaction between the gateway and CA-IDMS differs, depending on whether the gateway is running in the server address space or in a user address space. When running in the server address space, the interaction is by means of either the cross memory interface or the LU6.2 (APPC) API. When the gateway runs in a user address space, the interaction is by means of the batch CV facility (Mini-CV).

Gateway System Catalogs

Every relational database management system maintains catalogs to store information about its tables. The catalogs include data about:

- How the tables are organized
- Who owns the tables
- Where the tables are stored

The EDBC tools must encounter this information in the gateway system catalogs. The system catalogs consist primarily of views based on the CA-IDMS system tables. Other gateway system catalogs consist of tables that store EDBC objects such as reports or menus that have been created with the EDBC user interfaces.

The gateway system catalogs do not replicate the CA-IDMS system tables. As a result, these catalogs do not have to be updated each time the CA-IDMS system tables are updated. The system catalogs that store gateway-specific information are updated only by the gateway.

Chapter

4

Preparing for Installation

The installation process involves three components: the EDBC server, the CA-IDMS gateway, and the gateway user interfaces that run on OS/390.

This chapter:

- Outlines installation requirements for hardware, software, and storage.
- Describes what experience and authority is required of the installer.
- Addresses the key aspects of configuring the networking software to permit communication between a remote computer system and the OS/390 system where the gateway resides.
- Summarizes the major tasks that are required to install the components and to enable remote and local gateway users to access CA-IDMS Central Versions.

Before Beginning the Installation

Before you begin the installation, read this chapter to get an overview of the process. In addition, read the Release Notes for the current version of the EDBC server and the gateway. The Release Notes are provided in hard copy and can update the information in this guide.

Software Requirements

The CA-IDMS gateway requires the software resources listed in the following table:

Software	Specifications
IBM Operating System	Any currently supported release of MVS/ESA, OS/390, and z/OS
Application Subsystem	TSO/E
Communication Access Method: VTAM	ACF/VTAM
One of the following communication protocols: ■ SNA LU0 ■ SNA LU62 ■ KNET TCP/IP ■ IBM TCP/IP ■ CCI ■ SNS/TCP	
DB System:	CA-IDMS Release 12.0 and above with the SQL option.
Security Facilities	The IMS gateway supports any of the following security facilities: ■ IBM Resource Access Control Facility (RACF) ■ Computer Associates Access Control Facility 2 (CA-ACF2) with support for IBM's Security Access Facility (SAF) ■ Computer Associates Top Secret Security (CA-TSS)
User interfaces on client platform	Supported interfaces include: EDBC Client Ingres Client Jasmine Client The client can reside on Microsoft Windows or any of more than 30 UNIX platforms.

Software	Specifications
Gateway user interfaces on OS/390 platform	Netu SQL Terminal Monitor Embedded SQL (ESQL) Preprocessor for C ESQL for PL/I

Storage Requirements

The gateway requires the storage specified in the following sections.

Disk Storage

The gateway datasets require 290 cylinders of space on a 3380 or 3390 storage device.

If an OS/390 archive product such as CA-ASM2 or DFHSM is installed, exclude the gateway datasets from migration.

Virtual Storage Requirements

The gateway requires the virtual storage described in the following sections:

Gateway Address Space

The CA-IDMS gateway and server are installed together in a single OS/390 address space. This code runs almost entirely above the 16 MB line, except when it invokes OS/390 utilities. The gateway address space requires 640 bytes of Common System Area (CSA) storage below the 16 MB line for intra-address communication.

The server can support 240 simultaneous connections if the address space has a region size of 56 MB. The minimum region size is 16 MB, which supports 32 concurrent connections. A connection is considered a virtual gateway user on VMS, UNIX, or OS/390.

During initialization, the address space requires a working set of approximately 2100 KB. The initialization phase occurs only once, during startup, and takes about a minute. When dormant, the working set is approximately 300 KB.

When a new connection is established, it acquires a control block and about 220 KB of virtual storage above the line. This is allocated for the life of the connection and is returned when the user disconnects. An individual connection requires no additional storage in terms of Common System Area (CSA) storage below the 16 MB line.

The server address space in OS/390 runs as non-swappable code. If desired, this code can be loaded in the Link Pack Area (LPA) that OS/390 reserves for frequently used programs.

User Address Space

A user can access the gateway from OS/390, using the Terminal Monitor or one of the embedded OpenSQL preprocessors. When the gateway is accessed locally, it resides in the user address space along with the user interfaces. This requires a user region size of 4 to 6 MB.

Installer's Requirements

The CA-IDMS gateway requires that someone install it and provide ongoing support for the gateway users. The individual who assumes these responsibilities is referred to throughout this guide as the gateway database administrator (DBA).

Installing the CA-IDMS gateway requires a working knowledge of OS/390, TSO/E, ISPF, and JCL, as well as familiarity with CA-IDMS.

To expedite installation, the gateway DBA should have:

- Global system administrator status within OS/390, with authority to update the OS/390 system datasets listed in the following section, and to schedule or perform an IPL of OS/390.

  If necessary, the installation process can perform the required customization and place the files in gateway target libraries. Later, someone with the necessary authority can copy these files to system libraries.

- Ability to configure the software protocol for Net.

  This includes VTAM and either SNA LU0, SNA LU62, KNET TCP/IP, IBM TCP/IP, CCI, or SNS/TCP.

- Ability to create OS/390 user IDs and authorize them for TSO and CA-IDMS.

  These user IDs are assigned to the EDBC users who will access the gateway.

- CA-IDMS authority to create the sample databases that are part of the gateway installation and verification process.

Installer's Access to Resources

To install the gateway on OS/390, it is necessary to access or update a number of resources in OS/390 and CA-IDMS. These libraries and utilities are listed in the tables below. If you do not have the necessary authority, you can prepare the JCL and control statements and have the updates made by the appropriate individual.

The following table shows the OS/390 and CA-IDMS libraries that are referenced:

Libraries Referenced	Library Name	Access Mode
OS/390	SYS1.LINKLIB	Execute and optional update
	SYS1.MACLIB	Read
	SYS1.PROCLIB	Execute and optional update
	SYS1.SORTLIB	Execute
	SYS1.VTAMLIB	Optional update
	SYS1.VTAMLST	Optional update
	SYS1.SISTMAC1	
	SYS1.AMODGEN	
	SYS1.MODGEN	
CA-IDMS	CA-IDMS Installation Load Library	Execute

The following table shows the OS/390 and CA-IDMS utilities that are referenced:

System	Utility Name	Access Mode
OS/390	IDCAMS	Execute
	IEBUPTE	Execute
	IEFBR14	Execute
	IEV90/ASMA90	Execute
	IEWL	Execute
	IKJEFT01	Execute
CA-IDMS	IDMSBCF	Execute

Installation Overview

The installation procedure is modeled on a standard OS/390 installation procedure. The installation process uses TSO to customize JCL and control statements that are provided on the gateway product tape. The estimate of the time required to install the gateway is approximately eight hours.

The installation procedure is referred to as the EDBC Installation and Verification Procedure (IIVP). The IIVP installs the following:

- CA-IDMS gateway
- EDBC server
- Gateway user interfaces on OS/390 (the EDBC Terminal Monitor, the Embedded SQL Preprocessor for C, and the Embedded SQL Preprocessor for PL/I)

The installation process also creates sample CA-IDMS databases. These databases are provided so that the gateway DBA can verify that the gateway has been successfully installed. Operating locally under TSO, the gateway DBA issues SQL queries to the sample databases. The resulting data is presented just as it would appear to an EDBC end user.

Installation Summary

The installation of the gateway is completed in several phases as follows:

- Installation of the CA-IDMS gateway and EDBC server as described in the "Installing the Gateway" chapter
- Verification of the gateway installation as described in the "Installing the Gateway" chapter
- Network configuration as described in the "Configuring the EDBC Server" chapter
- Setting up gateway users as described in the "Maintaining the Gateway" chapter

Installing the Gateway

This section summarizes the procedure for installing the gateway components on OS/390. The steps in the installation phase are as follows:

1. Allocate and load the gateway datasets from the product tape (stage0).
2. Customize the macros in the file that contains the stage1 input.
3. Customize and submit the JCL to take the stage1 input file and produce the stage2 jobs.

4. Submit the stage2 jobs for execution, one at a time and check the return codes.
5. Define the EDBC subsystem to OS/390 and APF authorize the EDBC load libraries.
6. Perform a CA-IDMS sysgen to define the CA-IDMS resources (tasks, transactions, LINEs, PTERMs, and so on) necessary to interface the CA-IDMS gateway to the CA-IDMS Central Version. The sysgen macro statements are created during stage2 processing and are placed in the SAMPLE.CNTL library as member IDMSxxx (xxx = CV number).
7. Modify the TSO logon procedure used by the gateway DBA.

See the "Installing the Gateway" chapter for a detailed description of the installation procedure.

Verifying Gateway Functionality

Once installed, you can test the gateway by accessing the sample database that was created during the installation.

To verify the gateway installation, the following steps must be completed:

1. As the gateway DBA, log on to TSO using the modified TSO logon procedure.
2. Invoke the EDBC Terminal Monitor and test local access to the sample database.

See the "Installing the Gateway" chapter for a detailed description of the procedures to complete these steps.

Supporting Gateway Users

The gateway DBA must perform the following tasks to enable EDBC users to access the gateway:

- Create OS/390 user IDs for EDBC users
- Define EDBC users to the gateway
- Define and authorize EDBC users to the CA-IDMS CV

Chapter 5

Installing the Gateway

This chapter provides step-by-step instructions for installing the CA-IDMS gateway on OS/390. The installation procedure is referred to as the EDBC Installation and Verification Procedure (IIVP).

This installation phase is organized into four major stages:

- Customizing the stage1 input and producing the stage2 jobstream
- Submitting the stage2 jobstream jobs
- Completing the final installation procedures
- Verifying functionality and network connectivity

Many of the tasks involve customizing JCL and control statements provided on the gateway product tape. Most of the work is performed under TSO.

Installation Summary

The following table lists the steps you will perform during the installation. Each step is described in greater detail in the sections that follow this table.

	Task	Output
1.	Back up previous gateway installation, if necessary. See Backing up Previous Installation Datasets section.	Tape containing previous installation files and any applications stored in user interface catalogs.

	Task	Output
2.	Create and submit IEBGENER job to restore bootstrap JCL from tape. See Allocating and Loading the Installation Datasets section.	Stage 0 installation jobstream restored into data set with default name EDBC.V2R3.STAGE0.JOB.
3.	Edit and submit stage0 jobstream to allocate and copy gateway data sets onto disk. See Customizing the Stage1 Input section.	Gateway installation data sets, with default names such as: EDBC.V2R3.FILES.ASM EDBC.V2R3.FILES.MACLIB EDBC.V2R3.FRONT.LOAD
4.	Use ISPF to customize statements in stage1 input files to reflect installation-dependent values: ■ IGWFJOB ■ IGWFUSER ■ IGWFINET ■ IGWFIDMS ■ IGWFPSVR ■ IGWFPIPE ■ IGWFBLD See Customizing the stage1 Input section.	Customized and saved statements in member IGWFSTGS in data set with default name EDBC.V2R3.FILES.ASM.
5.	Customize and submit assembly JCL that uses stage1 input to produce stage2 jobstream. Input is member IGWFSTGS in data set with default name EDBC.V2R3. SAMPLE.CNTL. See the stage1 Jobstream.	Stage2 jobstream with report and listing.

	Task	Output
6.	Review and update if necessary jobs in IIVP stage2 jobstream, which is the output of the IIVP stage1. Located in data set with default name EDBC.V2R3.STAGE2.CNTL. Submit jobs one at a time in the following sequence: IGWFVPA0 IGWFVPS0 IGWFVPS1 IGWFVPS2 IGWFVPS3 IGWFVPS4 IGWFVPI0 IGWFVPN0 IGWFVPP0 IGWFVPZ9 See Submitting the Stage2 Jobstream Jobs section.	Customized files that build the gateway environment, creating load module, authorizing catalogued procedures, etc.
7.	Define the EDBC gateway subsystem to OS/390 and APF authorize EDBC load libraries.	Saved IEFSSNxx member and IEAAPFxx member of SYS1.PARMLIB.
8.	Customize TSO logon procedure for the gateway DBA.	Saved logon procedure.
9.	Perform a CA-IDMS sysgen to define CA-IDMS resources (programs, tasks, and so on)	Not applicable.

Installation Framework

The CA-IDMS gateway is defined to OS/390 as one subsystem. This subsystem is used by the CA-IDMS gateway and EDBC server. It has an associated address space, known as the gateway address space.

The overall gateway subsystem has the default name EDBC. It is recommended that you accept this default name, as it facilitates the installation.

You must IPL the operating system to define the gateway subsystem to OS/390 and permanently APF authorize the gateway load libraries. This can be done either before or after the installation process.

Read through this entire chapter to get an overview of the process before you begin the installation. In addition, be sure to read the Readme file for the current version of the product.

Installation Expectations

The IIVP stage2 jobstream accomplishes the following operations without an IPL:

- Customize the installation and run-time parameters that the gateway refers to as *logical symbols*.
- Create and initialize the sample CA-IDMS databases using the CA-IDMS gateway.
- Invoke the EDBC Terminal Monitor as a batch TSO application to verify access to the sample databases through the CA-IDMS gateway.
- Customize JCL for additional sample databases. This JCL can be submitted after the installation is complete.

The following operations are *not* performed by the IIVP stage2 jobstream:

- Adding JCL passwords to any jobs. Use an installation-approved method to provide this information.
- Adding the gateway load libraries to the APF list
- Defining the gateway subsystem to OS/390
- Updating the TSO logon procedure to access the gateway
- Installing the network software on the server and client. This must be done before the gateway server can be started.
- Performing a CA-IDMS sysgen to define new resources

The EDBC Product Tape

The EDBC product tape contains all the files needed to install the CA-IDMS gateway.

The EDBC server and CA-IDMS gateway are distributed on a single 9-track tape or cartridge. The tape has the following characteristics:

- IBM 3240 type or IBM 3480

- Standard label format, made using the IBM utilities IEBCOPY and IEBGENER
- Contains 40 files
- VOLSER printed on the external label of the tape (required information for installation procedure)
- Contents require approximately 270 cylinders of storage on a 3380 or 3390 DASD

Backing Up Previous Installation Datasets

To save the data sets from an existing installation, either back up the existing installation files, or assign a different data set prefix to the new installation files.

***Important**! If you assign the new gateway files the same dataset prefix as the existing files, the old files will be deleted before the new files are loaded.*

Allocating and Loading the Installation Datasets

The first file on the gateway product tape file is a bootstrap jobstream that prepares for the allocation and copying of the installation data sets. After the stage0 jobstream has been restored from File 1 of the tape, customize, and submit the jobstream for execution. The jobstream will allocate and load all the remaining installation data sets from tape to disk.

Creating JCL to Restore the Stage0 Jobstream

The following steps detail how to create a job, called IGWFJOB0, that uses the IEBGENER utility to allocate and copy File 1 from the product tape to a disk data set.

1. In an OS/390 data set, create JCL similar to the following sample:

```
//IGWFJOB0 JOB
//*
//* ALLOCATE AND COPY FILE 1 TO A DISK DATASET
//* THE RESTORED FILE IS THE STAGE 0 JOBSTREAM
//*
//COPYFILE    EXEC PGM=IEBGENER
//SYSIN       DD   DUMMY
//SYSPRINT    DD   SYSOUT=*
//SYSUT1      DD   DSN=EDBC.TV2R3.INSTALL.CNTL,
//                 VOL=(,RETAIN,SER=IDxxyy),
//                 UNIT=TAPE,LABEL=(1,SL),
//                 DISP=(OLD,KEEP)
//SYSUT2      DD   DSN=EDBC.V2R3.STAGE0.JOB,
//                 DISP=(NEW,CATLG,DELETE),
//                 UNIT=3380,VOL=SER=volser,
//                 SPACE=(TRK,(1,1))
```

2. Make sure the jobcard is valid.
3. For the SYSUT1 DD statement, specify the VOL parameter as listed on the gateway product tape.
4. For the SYSUT2 DD statement, specify:
   - The name under which to store the data set. The default is EDBC.V2R3.STAGE0.JOB.
   - The name of the physical unit on which the gateway data sets will be stored. The default is 3380.
   - The volser number of the disk to which this data set is to be allocated.
5. Submit the job for execution.
6. If you encounter errors restoring this jobstream, correct the errors and repeat until the SYSUT1 file is loaded successfully.

Customizing the Stage0 Installation Jobstream

The previous step restored the stage0 installation jobstream into a data set with the default name EDBC.V2R3.STAGE0.JOB. This step also requires the product tape as input and creates the gateway installation data sets.

Note: This jobstream assigns a common data set prefix for all installation data sets. The default value for this prefix is EDBC.V2R3. It is recommended that you use this default prefix, since it simplifies the installation process. The default data set prefix is used throughout this guide to refer to data set names.

The following table shows the steps executed by stage0 JCL:

Stepname	Description	RETCODE
STEP010	An IDCAMS step that deletes any previous installation data sets. Returns a code of 8 if there are no data sets to delete.	0 or 8
STEP020	Executes the ALLOC inline procedure to allocate the installation data sets.	0
STEP030 through STEP110	Copies the installation data sets from the distribution tape to disk.	0 or 4

The following steps describe how to use the ISPF editor to edit the stage0 jobstream and how to submit the job:

1. Add a valid jobcard.
2. Customize the following values (all values that must be customized have the form #*VALUE*):

#*PREFIX*	Change this string to the value chosen for the installation data set prefix. The prefix can be a 1-n level qualifier. For example, if you choose the default value, *EDBC.V2R3*, issue the ISPF editor command: **C '#PREFIX' 'EDBC.V2R3' ALL**
#*DVOL*	Change this string to the serial number of the volume where the installation data sets will be allocated. For example, if you choose the default value, *MVSVOL*, issue the command: **C '#DVOL' 'MVSVOL' ALL**
#*TAPE*	Change this string to the unit name for the IBM tape drive where the product tape is mounted. If you choose the default value, *TAPE*, issue the command: **C '#TAPE' 'TAPE' ALL**
#*TVOL*	Change this string to the serial number of the volume of the product tape. Refer to the external label of the tape to find the proper value. Issue the command: **C '#TVOL' 'IDVJ14' ALL**
#*ADSK*	Change this string to the unit name of the target disk pack on which the installation data sets will reside. If you choose the default value, *3380,* issue the command: **C '#ADSK' '3380' ALL**
#*TDSK*	Change this string to the unit name of the work disks. If you choose the default value, *SYSDA,* issue the command: **C '#TDSK' 'SYSDA' ALL**
#*MAXBLK*	Change this string to the value of the maximum block size for the gateway load libraries. The maximum value is 32760. If you choose the default value, *23476*, issue the command: **C '#MAXBLK' '23476' ALL**

3. Submit this job for execution.

   STEP010 returns a non-zero return code if there are no data sets to delete. If you encounter any errors, review the output and take corrective action.

When the job completes properly, the gateway installation data sets have been allocated and unloaded from the product tape.

Gateway Datasets That Are Allocated and Restored

The following gateway installation data sets are allocated by the stage0 job. These data sets are listed as they would be named with the default prefix:

```
EDBC.V2R3.BACK.LOAD
EDBC.V2R3.DOC.TEXT
EDBC.V2R3.FILES.ASM
EDBC.V2R3.FILES.CLIST
EDBC.V2R3.FILES.DBRMLIB
EDBC.V2R3.FILES.ENGLISH.FAST.MSG
EDBC.V2R3.FILES.ENGLISH.SLOW.MSG
EDBC.V2R3.FILES.H
EDBC.V2R3.FILES.HLP
EDBC.V2R3.FILES.IIPARM
EDBC.V2R3.FILES.ISPLLIB
EDBC.V2R3.FILES.MACLIB
EDBC.V2R3.FILES.NAME.IINAME.ALL
EDBC.V2R3.FILES.PROCLIB
EDBC.V2R3.FILES.RTIFORMS.FNX
EDBC.V2R3.FILES.SQL
EDBC.V2R3.FILES.STARTUP.DATA
EDBC.V2R3.FILES.USERS.DATA
EDBC.V2R3.FILES.UTEXE.DEF
EDBC.V2R3.FRONT.LOAD
EDBC.V2R3.OBJ
EDBC.V2R3.NET.LOAD
EDBC.V2R3.PLI.INCLIB
EDBC.V2R3.SAMPLE.ASM
EDBC.V2R3.SAMPLE.C
EDBC.V2R3.SAMPLE.CNTL
EDBC.V2R3.SAMPLE.COBOL
EDBC.V2R3.SAMPLE.H
EDBC.V2R3.SAMPLE.PLI
EDBC.V2R3.SAMPLE.SASC
EDBC.V2R3.STAGE2.JOBS
EDBC.V2R3.STAGE2.CNTL
EDBC.V2R3.TEST.PLI
EDBC.V2R3.TEST.QPLI
EDBC.V2R3.TEST.SPLI
EDBC.V2R3.USER.LOAD
EDBC.V2R3.USER.CNTL
EDBC.V2R3.USER.OBJ
EDBC.V2R3.WS.CLIB
EDBC.V2R3.WS.H
EDBC.V2R3.WS.LOAD
EDBC.V2R3.FILES.NAME.IINAME.INIT
```

Note: The FILES.CLIST dataset is allocated with DCB=(RECFM=FB,LRECL=80,BLKSIZE=6160). If these attributes do not conform to your installation standards, use ISPF or some other utility to copy the FILES.CLIST members to a library allocated with the correct DCB attributes.

Customizing the Stage1 Input

The IIVP customizes the stage1 input, which consists of the statements listed in the following table. They are stored in the IGWFSTGS member of the EDBC.V2R3.FILES.ASM data set. Review each statement and, if necessary, use ISPF to customize it to reflect site-specific values.

Note: Do not modify the macros in EDBC.V2R3.FILES.MACLIB.

Statement	Purpose
IGWFJOB	Customizes jobcard parameters
IGWFUSER	Customizes initial gateway users
IGWFINET	Customizes the EDBC server
IGWFIDMS	Customizes the CA-IDMS gateway
IGWFPSVR	Customizes the protocol server
IGWFPIPE	Customizes the CA-IDMS LU62 interface (PIPE)
IGWFBLD	Builds the IIVP jobstream

Note: The first statement must be IGWFJOB, and the last statement must be IGWFBLD.

If you specify a parameter with special characters, enclose them in quotes. If you provide a NULL value, do *not* place it in quotes.

The following sections define each statement, and outline its parameters, usage, and defaults.

IGWFJOB Statement: Specifying Jobcard Parameters

The IGWFJOB statement defines the installation-specific values that are used to build the JCL job statements for the IIVP stage2 jobstreams. You must specify this statement first.

This statement is defined as follows:

```
IGWFJOB ACCT='(ACCT,INFO)',         ACCOUNTING INFO     X
       CLASS=A,                     EXECUTION CLASS     X
       JOBNAME='IGWFVP',            JOBNAME PREFIX      X
       NOTIFY=,                     NOTIFY USER         X
       MSGCLASS=H,                  MSGCLASS            X
       MSGLEVEL='(1,1)',            MSGLEVEL            X
       PGMRNAME='EDBC INSTALL',     PROGRAMMER NAME     X
       REGION=4096K,                REGION SIZE         X
       SYSAFF=,                     SYSTEM AFFINITIES   X
       USER=EDBCDBA                 GATEWAY DBA
```

Most of these are standard OS/390 parameters; therefore, enter values that comply with the conventions at your site. The following table lists only parameters with special significance for the installation procedure:

IGWFJOB Parameter	Usage	Default
ACCT=	The information used to build the accounting information in the IIVP jobstream job statements. This information is placed in parenthesis on each job statement generated by the IIVP stage1.	' '
CLASS=	The CLASS= parameter on the IIVP jobstream job statements.	*A*
JOBNAME=	Prefix for each job name in the IIVP stage2 jobstream. The IIVP jobstream appends a two-character suffix to this supplied value. The prefix is limited to a maximum of six characters. If it exceeds this limit, the prefix is truncated and a warning message is issued.	*IGWFVP*
NOTIFY=	The value assigned to the NOTIFY= parameter of the IIVP job statements.	IGWFJOB USER= parameter
MSGCLASS=	The MSGCLASS= parameter on the IIVP jobstream job statements.	*H*
MSGLEVEL=	The MSGLEVEL= parameter on the IIVP jobstream job statements.	*(1,1)*
PGMRNAME=	The programmer name field that appears in each job of the IIVP jobstream. Limited to a maximum of 20 characters. If the value specified exceeds this maximum, it is truncated and a warning message is issued.	*'EDBC INSTALL'*
REGION=	The value assigned to the REGION= parameter of the IIVP JOB STATEMENTS.	*0M*

IGWFJOB Parameter	Usage	Default
USER=	Enter the TSO user ID for the gateway DBA. This value is added automatically to the USERS= list of gateway users in the IGWFPARM statement. It is also used as the default value for the NOTIFY= parameter in this statement.	*EDBCDBA*
SYSAFF=	Used to generate a /*JOBPARM SYSAFF= statement for generated stage2 jobs. The value specified indicates the systems in a JES2 multi-access spool configuration that are eligible to process stage2 jobs. See the JCL Reference guide for acceptable values.	*ANY*

IGWFUSER Statement: Specifying Initial Gateway Users

The IGWFUSER statement specifies the initial users that are to be defined to the gateway. The information specified in this statement is used to update the EDBC.V2R3.FILES.USERS.DATA data set and creates the user entries for the EDBC system catalogs.

This statement is specified once for each user. It is defined as follows:

```
IGWFUSER TYPE=IDMS,           IDMS USER_NAME        X
         USER_NAME=EDCUSR1,                         X
         DBA_NAME=,           DEFAULT DBA_NAME      X
         SCHEMA=              DEFAULT SCHEMA
```

The following table describes the parameters in the IGWFUSER statement:

IGWFUSER Parameter	Usage	Default
TYPE=	The server type name. This must be IDMS.	None
USER_NAME=	The remote authid of the gateway user.	None
DBA_NAME=	The DBA name associated with this user.	Value of USER_NAME
SCHEMA	The schema name associated with this user.	Value of USER_NAME=

IGWFINET Statement: Specifying EDBC Parameters

The IGWFINET statement specifies the values for initializing and configuring EDBC. This statement is defined as follows:

```
IGWFINET  SUBSYS=EDBC,       SUBSYSTEM NAME          X
          ID=I1,             INSTALLATION ID         X
          SVRTYPE=IDMS,      DEFAULT SERVER TYPE     X
          SECURITY=NONE,     DEFAULT SECURITY        X
          TIMEOUT=,          GATEWAY WAIT TIMEOUT    X
          MAXUSER=64,        MAXIMUM USERS           X
          NETLMOD=IIPSERV,   NETWORK LOAD MODULE     X
          TIMEZONE=7         TIMEZONE
```

Customize the parameters for the IGWFINET statement as shown in the following table:

IGWFINET Parameter	Usage	Default
SUBSYS=	Four-character EDBC subsystem ID. Value must be added to the IEFSSNxx member in SYS1.PARMLIB. For more information, see Defining the gateway Subsystem to OS/390 section. If values are specified for DBNM and ISVR in the previous statement, they must match this value.	*EDBC*
ID=	Two-character installation code for the gateway installation. The value for this parameter must match the value for the INSTALL= parameter in the IGWFPSVR statement. The installation code is assigned to the following installation data sets (shown with default names): EDBC.V2R3.FILES.NAME.IICOMSVR.I1 EDBC.V2R3.FILES.NAME.IIDB2.I1 EDBC.V2R3.FILES.NAME.IIIDMS.I1 EDBC.V2R3.FILES.NAME.IIEDBC.I1 EDBC.V2R3.FILES.NAME.IILOGIN.I1 EDBC.V2R3.FILES.NAME.IINODE.I1 EDBC.V2R3.FILES.NAME.IIIMS.I1 EDBC.V2R3.FILES.NAME.IIVSAM.I1 EDBC.V2R3.FILES.NAME.IIVANT.I1	*I1*
SVRTYPE=	Default server class, as specified when accessing CA-IDMS from TSO or a batch application. **Note:** This value must be entered as shown.	*IDMS*

IGWFINET Parameter	Usage	Default
SECURITY=	Type of security package to which the gateway interfaces. The options are: ■ *RACF* ■ *ACF2* ■ *TSS* ■ *None* The SECURITY= parameter is used to set the logical symbol II_SECURITY= to the specified value.	*NONE*
TIMEOUT=	Specifies the maximum time, in minutes, that a gateway thread waits before it is disconnected. The range for this parameter is between 0 and 9999. A 0 value indicates that no wait time limits will be enforced. The TIMEOUT= parameter is used to set the II_INACTV_TMOUTINT logical symbol. This logical, in turn, is used to compute the INACTIVE INTERVAL value (stall interval) for the EDBCRSPD cross memory task using the following computation: INACTIVE INTERVAL = II_INACT_TMOUTINT + II_FORCE_TMOUTINT + 1 minute If either logical is increased, the stall interval for the EDBCRSPD task should be adjusted accordingly. The EDBCRSPD task must never terminate before the CA-IDMS gateway thread.	*0*
MAXUSER=	Maximum number of remote users that can connect to EDBC. **Note:** This parameter can be set to a value up to 4080 for the current release.	*64*
NETLMOD=	Name of the network load module to use for EDBC initialization. The NETLMOD= parameter is used to set the II_NET_LMOD logical symbol to the specified value.	IIPSERV
TIMEZONE=	Specifies the difference in hours between Greenwich mean time (GMT) and the local time zone where the gateway server is running.	*0*

IGWFIDMS Statement: Specifying CA-IDMS Gateway Parameters

The IGWFIDMS statement is defined as follows:

- Site-specific values for the CA-IDMS data sets and the CA-IDMS Central Version
- The location for the sample CA-IDMS database created as part of the installation process

This statement is defined as follows:

```
IGWFIDMS   INSTALL=UPGRADE,            INSTALL TYPE               X
           IDMS_LOADLIB='idms.load',   CA-IDMS loadlib            X
           IDMS_CVNUM='                Target CV # of install     X
           IDMS_DCMSG='idmsdcmsg',     CA-IDMS DCMSG              X
           IDMS_SECSGON='              Cross memory signon option X
           IDMS_SYSCTL='sysctl',       CA-IDMS SYSCTL             X
           IDMS_DISTSRC='distsrc',     CA-IDMS DISTSRC            X
           IDMS_MACLIB='maclib',       CA-IDMS MACLIB             X
           IDMS_OBJ='objlib',          CA-IDMS OBJLIB             X
           HLQ_PREFIX='hlqprefix',     Dataset prefix             X
           HOT_CONNECT=,               Hot Connect Option         X
           DASD_UNIT=,                 DASD UNIT                  X
           DBPROCS_MAX=128,            MAX CONCURRENT DBPROCS     X
           DBPROCS_MEMORY=1024,        MAX MEMORY FOR DBPROCS     X
           IIDMCL_BUFNAME=,            Buffer Name                X
           MBUF_PGSIZ=,                Largest Page Size          X
           IBUF_PAGES=,                # of Buffer pages          X
           MBUF_PAGES=,                Max Buffer pages           X
           CDMSLIB=,                   CA-IDMS CDMSLIB            X
           CVDMCL_NAME=,               CV Global   DMCL.          X
           DBNAME_TABLE=,              CV DBNAME Table            X
           DBA_LOADLIB=,               IDMS DBA load lib          X
           SYSTEM_DBNAME=,             CV System DBNAME           X
           SYSTEM_OWNER=               EDBC catalog owner         X
           SECONDARY_CATALOG=,         NEW or OLD                 X
           SQL_DBNAME=,                GATEWAY DBNAME             X
           SQLSEG_NAME=,               GATEWAY SEGMENT name       X
           CATSEG_NAME=,               STDCAT "                   X
           DEFSEG_NAME=,               DEFAULT SEGMENT            X
           DEFSCH_NAME=,               DEFAULT SCHEMA             X
           SQLAREA_LOPAGE=,            DDLCAT    lopage           X
           SQLAREA_PAGES=,                       pages            X
           SQLAREA_PGSIZ=,                       page size        X
           SQLAREA_DDNAME=,                      DDname           X
           SQLINDX_LOPAGE=,            DDLCATX   lopage           X
           SQLINDX_PAGES=,                       pages            X
           SQLINDX_PGSIZ=,                       page size        X
           SQLINDX_DDNAME=,                      DDname           X
           SQLLOAD_LOPAGE=,            DDLCATL   lopage           X
           SQLLOAD_PAGES=,                       pages            X
           SQLLOAD_PGSIZ=,                       page size        X
           SQLLOAD_DDNAME=,                      DDname           X
           CATINDX_LOPAGE=,            CINDEX    lopage           X
           CATINDX_PAGES=,                       pages            X
           CATINDX_PGSIZ=,                       page size        X
           CATINDX_DDNAME=,                      DDname           X
           CATAREA_LOPAGE=,            STANDARD  lopage           X
           CATAREA_PAGES=,                       pages            X
           CATAREA_PGSIZ=,                       page size        X
           CATAREA_DDNAME=,                      DDname           X
           EXTINDX_LOPAGE=,            EINDEX    lopage           X
           EXTINDX_PAGES=,                       pages            X
           EXTINDX_PGSIZ=,                       page size        X
           EXTINDX_DDNAME=,                      DDname           X
           EXTAREA_LOPAGE=,            EXTENDED  lopage           X
           EXTAREA_PAGES=,                       pages            X
           EXTAREA_PGSIZ=,                       page size        X
           EXTAREA_DDNAME=,                      DDname           X
           DEFINDX_LOPAGE=,            DINDEX    lopage           X
           DEFINDX_PAGES=,                       pages            X
           DEFINDX_PGSIZ=,                       page size        X
```

```
DEFINDX_DDNAME=,                      DDname          X
DEFAREA_LOPAGE=,           DDATA      lopage          X
DEFAREA_PAGES=,                       pages           X
DEFAREA_PGSIZ=,                       page size       X
DEFAREA_DDNAME=,                      DDname          X
VOLSER=                               VOLSER
```

The following table describes the parameters in the IGWFIDMS statement:

IGWFIDMS Parameters	Usage	Default
INSTALL=	Specifies whether the gateway is being installed for first time (NEW) or if the gateway has been previously installed (UPGRADE). If NEW is used, existing gateway catalogs are deleted and new catalogs are created. If UPGRADE is used, existing gateway catalogs are preserved.	*UPGRADE*
IDMS_LOADLIB=	The DSN of the CA-IDMS installation load library.	
IDMS_DCMSG=	The DSN of the CA-IDMS installation DC Message area, used in local mode processing for file formatting.	
IDMS_SECSGON	The CA-IDMS cross memory connect security signon option. YES--Validate the user and password. The connect is aborted if validation fails. NO--Do not validate the user or password. ***Caution***: *The user inherits the security profile associated with the CA-IDMS task id EDBCxxxx.* USER--Validate the user but not the password. ***Caution***: *The user must be defined to CA-IDMS with PASSWORD NOT ASSIGNED attribute.*	*YES*

IGWFIDMS Parameters	**Usage**	**Default**
IDMS_SYSCTL=	The DSN of the CA-IDMS SYSCTL for the target CV of this install. A single instance of the EDBC server can communicate with multiple Central Versions. The installation of EDBC requires modification of a CV's DMCL and DBNAME table, and therefore a SYSCTL dd statement, to process the modifications under the CV's control.	
IDMS_CVNUM=	The central verion number for the target CV of this install.	*NONE*
IDMS_DISTSRC=	The DSN of the CA-IDMS installation distribution source library (contains the members, TABLEDDL and VIEWDDL).	
IDMS_MACLIB=	The DSN of the CA-IDMS installation macro library.	
IDMS_OBJLIB=	The DSN of the CA-IDMS installation object library.	
HLQ_PREFIX=	The High Level Qualifier for all data sets to be created.	
HOT_CONNECT=	Specifies whether a hot connection to CA-IDMS should be established during EDBC server initialization. The HOT_CONNECT parameter is used to set the logical symbol IDMS_HOT_CONNECT to the specified value.	*YES*
DASD_UNIT=	The default unit specification for DASD.	*SYSDA*
DBPROCS_MAX=	Specifies the maximum number of DBPROCS that can be run by a gateway user.	128
DBPROCS_MEMORY=	Specifies the maximum amount of storage (K) that can be allocated by a DBPROC.	1024
IIDMCL_BUFNAME=	The buffer name to be used for all gateway areas.	*II_BUFFER*
MBUF_PGSIZ=	The largest page size specified for any gateway area. This is used in the DMCL buffer statement	*4276*

IGWFIDMS Parameters	**Usage**	**Default**
IBUF_PAGES=	The initial number of buffer pages to be specified in the DMCL buffer statement.	*200*
MBUF_PAGES=	The maximum number of buffer pages to be specified in the DMCL buffer statement.	*1000*
CDMSLIB=	Supplies the name of the CA-IDMS CDMSLIB library that will contain the new DMCL and DBNAME tables generated during the stage2 process.	None
CVDMCL_NAME=	The name of your CV Global DMCL.	
DBNAME_TABLE=	The name of your CV Database Name Table.	
DBA_LOADLIB=	The DSN of your CA-IDMS DBA load library. This is used for local mode processing, it contains Global DMCL and DBNAME table.	
SYSTEM_DBNAME=	The CV System DBNAME.	*SYSTEM*
SYSTEM_OWNER=	The schema name that will be the qualifier for all the EDBC catalogs.	*$EDBC*
SECONDARY_CATALOG=	NEW allocates a new secondary catalog. OLD uses an existing secondary catalog. NEW is typically recommended for the initial EDBC implementaion. This ensures that EDBC metadata and catalogs are maintained independently of existing CA-IDMS database areas. OLD is used when there is an existing secondrary catalog (DDLCAT) that contains SQL schemas defined for NETWORK schemas. **Note:** Installing EDBC metadata and catalogs into SYSSQL is NOT RECOMMENDED.	*NEW*
SQL_DBNAME=	The DBNAME (secondary catalog) where the gateway catalogs will be created. When SECONDARY_CATALOG=NEW is specified, this value will be used as the name of the new secondary catalog.	*EDCSQL*
SQLSEG_NAME=	The segment name of the new gateway catalog.	*EDCSQL*

IGWFIDMS Parameters	Usage	Default
CATSEG_NAME=	The segment name of the new gateway Standard Table Database Area(s).	*EDBC*
DEFSEQ_NAME=	The segment name of the new gateway Default Database Area.	*EDBCDBA*
DEFSCH_NAME=	The SCHEMA name of the new gateway Default Database Area.	*$EDBCDBA*
SQLAREA_LOPAGE=	The low page number of the gateway SQL Catalog DDLCAT Area.	*100001*
SQLAREA_PAGES=	The number of pages for the gateway SQL Catalog DDLCAT Area.	*300*
SQLAREA_PGSIZ=	The page size of the gateway SQL Catalog DDLCAT Area.	*5492*
SQLAREA_DDNAME=	The DDname of the gateway SQL Catalog DDLCAT Area.	*IISQLF1*
SQLINDX_LOPAGE=	The low page number of the gateway SQL Catalog DDLCATX Area.	*100301*
SQLINDX_PAGES=	The number of pages for the gateway SQL Catalog DDLCATX Area.	*100*
SQLINDX_PGSIZ=	The page size of the gateway SQL Catalog DDLCATX Area.	*5492*
SQLINDX_DDNAME=	The DDname of the gateway SQL Catalog DDLCATX Area.	*IISQLF2*
SQLLOAD_LOPAGE=	The low page of the gateway SQL Catalog DDLCATL Area.	*100401*
SQLLOAD_PAGES=	The number of pages for the gateway SQL Catalog DDLCATL Area.	*50*
SQLLOAD_PGSIZ=	The page size of the gateway SQL Catalog DDLCATL Area.	*5492*
SQLLOAD_DDNAME=	The DDname of the gateway SQL Catalog DDLCATL Area.	*IISQLF3*
CATINDX_LOPAGE=	The low page number of the gateway Standard Table CINDEX Area.	*101001*
CATINDX_PAGES=	The number of pages for the gateway Standard Table CINDEX Area.	*100*

IGWFIDMS Parameters	Usage	Default
CATINDX_PGSIZ=	The page size of the gateway Standard Table CINDEX Area.	*4276*
CATINDX_DDNAME=	The DDname of the gateway Standard Table CINDEX Area.	*IICATF1*
CATAREA_LOPAGE=	The low page number of the gateway Standard Table CATALOG area.	*101101*
CATAREA_PAGES=	The number of pages for the gateway Standard Table CATALOG Area.	*500*
CATAREA_PGSIZ=	The page size of the gateway Standard Table CATALOG Area.	*4276*
CATAREA_DDNAME=	The DDname of the gateway Standard Table CATALOG Area.	*IICATF2*
EXTINDX_LOPAGE=	The low page number of the gateway Standard Table EINDEX Area.	*102001*
EXTINDX_PAGES=	The number of pages for the gateway Standard Table EINDEX Area.	*100*
EXTINDX_PGSIZ=	The page size of the gateway Standard Table EINDEX Area.	*4276*
EXTINDX_DDNAME=	The DDname of the gateway Standard Table EINDEX Area.	*IICATF3*
EXTAREA_LOPAGE=	The low page number of the gateway Standard Table EXTENDED Area.	*102101*
EXTAREA_PAGES=	The number of pages for the gateway Standard Table EXTENDED Area.	*500*
EXTAREA_PGSIZ=	The page size of the gateway Standard Table EXTENDED Area.	*4276*
EXTAREA_DDNAME=	The DDname of the gateway Standard Table EXTENDED Area.	*IICATF4*
DEFINDX_LOPAGE=	The low page number of the gateway Standard Table DINDEX Area.	*103001*
DEFINDX_PAGES=	The number of pages for the gateway Standard Table DINDEX Area.	*200*
DEFINDX_PGSIZ=	The page size of the gateway Standard Table DINDEX Area.	*4276*

IGWFIDMS Parameters	Usage	Default
DEFINDX_DDNAME=	The DDname of the gateway Standard Table DINDEX Area.	*IIDBAF1*
DEFAREA_LOPAGE=	The low page number of the gateway Standard Table DDATA Area.	*103201*
DEFAREA_PAGES=	The number of pages for the gateway Standard Table DDATA Area.	*800*
DFAREA_PGSIZ=	The page size of the gateway Standard Table DDATA Area.	*4276*
DEFAREA_DDNAME=	The DDname of the gateway Standard Table DDATA Area.	*IIDAF2*
STORCLAS=	SMS storage class to use for the allocation of the CA-IDMS secondary catalog and EDBC areas.	*DASD_UNIT=* *VOLSER=*
VOLSER=	The VOLSER for gateway database area creation.	

IGWFPSVR Statement: Specifying Protocol Server Parameters

The IGWFPSVR statement specifies the network parameters for the EDBC server network interfaces: CCI, SNA LU0, SNA LU62, IBM TCP/IP, SNS/TCP, and KNET TCP/IP. This statement is coded, assembled, and linkedited into load module form during the installation process.

See the Install and Configure Communication Interfaces section of the "Configuring the EDBC Server" chapter for additional information.

The IGWFPSVR statement can be repeated up to 32 times for each protocol server type.

The IGWFPSVR statement is defined as follows:

```
IGWFPSVR TYPE=SNA_LU0,         TYPE OF SERVER           X
         MODETAB='EDCMODE',    MODE TABLE               X
         DLOGMOD='EDCLU0',     DEFAULT LOGON MODE       X
         ACB='EDCACBI1',       ACB NAME                 X
         RUSIZE=4096,          MAXIMUM RU SIZE          X
         PLU=,                 KNET PLU NAME            X
         PORT='134',           PORT ADDRESS             X
         INSTALL=I1,           INSTALLATION CODE        X
         PASS=' ',             OPTIONAL PASSWORD        X
         NODENAME=,            LOGICAL UNIT NAME        X
         USERID='TCPIP',       IBM TCP/IP USERID        X
         ENABLE=YES,           GENERATE PROTOCOL PARMS  X
         APPLID=,              APPLICATION ID - SNS/TCP X
         SYSID=,               SNS/TCP SUBSYSTEM ID     X
         PRODID=               CCI SUBSYSTEM ID
```

Customize the parameters for the IGWFPSVR statement as shown in the following table. If you do not enter a value for a parameter, it defaults to the value shown in the table. The system ignores values for non-selected protocols.

IGWFPSVR Parameter	Usage	Default
TYPE=	Type of protocol server. The options are: ■ *SNA_LU0* for SNA LU0 ■ *SNA_LU62* for SNA LU6.2 ■ *TCP_KNET* for KNET TCP/IP ■ *TCP_IBM* for IBM TCP/IP ■ *TCP_SNS* for SNS/TCP ■ *CCI* for CAI CCI	*SNA_LU0*
MODETAB=	SNA LU0 or SNA LU6.2 only: Name of the VTAM mode table that contains the default logon mode entry for this protocol server. Used to build the APPL statement and create the VTAM logon mode table for this protocol server.	*EDCMODE*
DLOGMOD=	SNA LU0 or SNA LU6.2 only: In conjunction with the MODETAB= parameter, which defines the default logon mode to be used in establishing sessions with this application.	*EDCLU0*

IGWFPSVR Parameter	**Usage**	**Default**
ACB=	SNA LU0 or SNA LU6.2 only: Name of the VTAM Access Control Block (ACB) used by the protocol servers. Also used to create the APPL definition for SNA protocol servers. This value is required to initialize the protocol interfaces. The value should not exceed eight characters. It must begin with an alpha or national character.	*EDCACBI1*
RUSIZE=	Sets the maximum RU size for the protocol servers. This value should be consistent with your installation's SNA standards. The default value is recommended.	*4096*
PLU=	KNET TCP/IP only: This value is the VTAM ACBNAME used by the KNET address space. The server must have access to the KNET runtime library either through a STEPLIB or LNKLST specification.	*None*
PORT=	All TCP/IP protocols: Value assigned for the port ID that is used to accept connections from remote gateway users. The port ID is an arbitrary value. Generally, avoid numbers lower than 1024 as these tend to be reserved for well-known TCP and UDP services. If you have more than one EDBC installation within the same OS/390 system, specify a unique port ID for each installation.	*134*
INSTALL=	Two-character installation code for the EDBC server. If unspecified, defaults to the value assigned to the IGWFINET ID= parameter (see the IGWFINET Statement: Specifying EDBC Parameters section).	*I1*
PASS=	Password.	None

IGWFPSVR Parameter	Usage	Default
NODENAME=	SNA_LU0 and SNA_LU62 only: Specifies the VTAM logical unit name for the associated ACB name. This parameter allows the LU name to differ from the ACB name. If NODENAME= is not specified, it defaults to the value of the ACB= parameter. If NODENAME= is specified and ACB= is not specified, ACB= defaults to the NODENAME= value. This parameter provides the label for the VTAM APPL statement generated by the IIVP stage2 jobstream.	ACB= parameter
USERID=	IBM TCP/IP only: This is the user ID of the IBM TCP/IP address space. The statement default value is TCPIP, which is also the default when installing IBM TCP/IP. Check with the system programmer to determine if a value other than this default was used.	*TCPIP*
ENABLE=	Specifies whether the IGWFPSVR statement will be processed or ignored during stage1. This parameter allows all protocol servers to be defined without enabling them. The options are: ■ NO IGWFPSVR statement is processed but the protocol server is not enabled during the gateway initialization. During stage1, results in a return code of 04. ■ YES IGWFPSVR statement is processed. The protocol server is enabled during the gateway initialization. All protocol initialization parameters are placed in one load module in EDBC.V2R3.NET.LOAD (IIPSERV). A member that can recreate this load module is customized in the IIVP stage2 jobstream. It is located in EDBC.V2R3.SAMPLE.CNTL (ASMPSERV).	*YES*
APPLID=	SNS/TCP only: Specifies the application program name that is passed to SNS/TCP. This name is used with the PASSWORD= value to authorize access to SNS/TCP. The name, if coded, must be an alphanumeric string up to 8 characters long.	Nulls

IGWFPSVR Parameter	Usage	Default
SYSID=	SNS/TCP only: Specifies the name of the SNS/TCP subsystem that is to be invoked by the TCP_SNS protocol server. The subsystem name is an alphanumeric string up to 4 characters long.	ACSS
PRODID=	CCI only: Applies only to the CCI system that is running. It must be a unique 20 character maximum Product ID and unique within the CCI system.	

IGWFPIPE Statement: Specifying CA-IDMS LU62 Interface Parameters

The IGWFPIPE statement should only be specified if the CA-IDMS gateway is to use the LU62 line driver. If the cross memory service interface is to be used, this statement and its parameters should be deleted from the stage1 input stream.

Note: It is strongly recommended that the cross memory service interface be used instead of the LU6.2 line driver. The cross memory interface provides many advantages over the LU6.2 technology, including performance and functional improvements as well as ease of installation.

The IGWFPIPE statement specifies the VTAM parameters for interfacing the gateway to the CA-IDMS address space. This statement is coded, assembled, and linkedited as module IIPIPE during the installation process. The jobstream which assembles and linkedits IIPIPE is saved in the SAMPLE.CNTL library so users can re-create it in the event of network changes.

Note: In addition to creating load module IIPIPE, this statement is used to generate VTAM APPL and MODEENT macro statements during the IIVP stage1 processing. In stage2, the APPL statements are added as major node PIPE to the VTAMLST library. The MODEENT statements are assembled and linkedited into VTAMLIB library. The stage2 processing noted above are performed by the stage2 job with the suffix P0. To insure that the stage2 processing conforms to installation standards, the gateway installer should consult with network and/or systems personnel before running the P0 job.

The IGWFPIPE statement parameters are as follows:

```
IGWFPIPE TYPE=IDMS,                     DBMS FOR PIPE            X
         INSTALL=,                      INSTALLATION ID          X
         EDBC_ACB=EDCALU62,             EDBC SERVER PIPE ACB     X
         EDBC_ACBPASS=,                 EDBC SERVER ACB PASSWORD X
         EDBC_NODENAME=NODELU62,        EDBC SERVER ACB NODENAME X
         IDMS_ACB=IDMSLU62,             IDMS LU62 DRIVER ACB     X
```

```
IDMS_ACBPASS=,                  IDMS LU62 ACB PASSWORD   X
IDMS_NODENAME=IDMSPIPE,         IDMS LU62 ACB NODENAME   X
IDMS_LINE=SNALU62,              IDMS LU62 LINE NAME      X
IDMS_PTERM=PTVTU,               IDMS LU62 PTERM PREFIX   X
IDMS_LTERM=LTVTU,               IDMS LU62 LTERM PREFIX   X
IDMS_SIGNON=YES,                LU62 SIGNON OPTION       X
IDMS_TASKID=RSPD,               IDMS TASK-CODE           X
DLOGMODE=EDCLU62,               DEFAULT LOGMODE          X
MODETAB=EDCMODE,                LOGMODE TABLE            X
NUMLUS=64,                      IDMS LOGICAL TERMINALS   X
RUSIZE=4096,                    RUSIZE                   X
ENABLE=YES                      ENABLE AT STARTUP
```

Customize the parameters for the IGWFPIPE statement as shown in the following table. If you do not enter a value for a parameter, it defaults to the value shown in the following table:

Parameters	Usage	Default
TYPE=	DBMS of LU62 interface. Currently the only option is CA-IDMS.	*IDMS*
EDBC_ACB=	Name of the VTAM Access Control Block (ACB) to be used by the EDBC server.	*EDCALU62*
EDBC_PASS=	EDBC server ACB password.	*None*
EDBC_NODENAME=	Specifies the VTAM logical unit name for the EDBC server ACB. This parameter allows the LU name to differ from the ACB name. This parameter provides the label for the VTAM APPL statement generated by the IIVP stage2 jobstream.	*EDBC_ACB=parameter*
IDMS_ACB=	Name of the VTAM Access Control Block (ACB) to be used by the CA-IDMS LU62 line driver.	*IDMSPIPE*
IDMS_ACBPASS=	CA-IDMS ACB password.	*None*
IDMS_NODENAME=	Specifies the VTAM logical unit name forIDMS_ACB= the associated IDMS ABC. This parameter allows the LU name to differ from the ACB name. This parameter provides the label for the VTAM APPL statement generated by the IIVP stage2 jobstream.	*IDMS ACB.parameter*

Parameters	Usage	Default
IDMS_LINE=	Specifies the name to be given to the CA-IDMS LU62 line. Used to create LINE macro definition for the CA-IDMS sysgen.	*SNALU62*
IDMS_PTERM=	Specifies the prefix value to use for naming the LU62 driver physical terminals. Used to create PTERM macro definitions for the CA-IDMS sysgen. Maximum of 5 characters.	*PTVTU*
IDMS_LTERM=	Specifies the prefix value to use for naming the LU62 driver logical terminals. Used to create LTERM macro definitions for the CA-IDMS sysgen. Maximum of 5 characters.	*LTVTU*
IDMS_TASKID=	Specifies the task-code to use to invoke the gateway transaction in the CA-IDMS CV. Maximum of 4 characters.	*RSPD*
IDMS_SIGNON=	Specifies whether the remote user name is to be passed to CA-IDMS during connection processing.	*YES*
DLOGMODE=	In conjunction with the MODETAB= parameter, defines the default logon mode entry to be used for establishing LU62 sessions. This parameter sets the DLOGMODE= value for the VTAM APPL statement generated by the IIVP stage2 jobstream.	*EDCMODE*
MODETAB=	Name of the VTAM mode table that contains the default logon mode entry for the LU62 session.	*PIPEMODE*
NUMLUS=	Name of the VTAM mode table that contains the default logon mode entry for the LU62 session.	*64*

Parameters	Usage	Default
RUSIZE=	Sets the maximum RU size for the LU62 PIPE. This value should be consistent with your installation's SNA standards. The default value is recommended. This parameter sets the RUSIZES= value for the VTAM MODEENT statement generated by the IIVP stage2 jobstream.	*4096*
ENABLE=	Specifies whether the CA-IDMS LU62 PIPE is to be activated during EDBC server startup.	*YES*

IGWFPIPE Statement: Specifying EXCI Interface Parameters

The IGWFPIPE statement specifies the EXCI parameters required for interfacing the gateway to the CICS address space. This statement generates the logical symbols, II_EXCI_APPL_ID and II_EXCI_NET_NAME. These logicals are added to IIPARM during stage2 processing.

The IGWFPIPE statement is as follows:

```
IGWFPIPE   TYPE=EXCI,      TYPE OF PIPE            X
           APPLID=,        APPLICATION ID          X
           NETNAME=        CONNECTION NETNAME
```

Customize the parameters for the IGWFPIPE statement as shown in the following table. If you do not enter a value for a parameter, it defaults to the value show in the following table:

IGWFPIPE Parameter	Usage	Default
TYPE=	Specifies the type of pipe interface. **Note:** This value must be entered as in the above statement example.	EXCI

IGWFPIPE Parameter	Usage	Default
APPLID=	Specifies the APPLID of the CICS system to connect to. The gateway passes this parameter as the value of the CICS_applid for the ALLOCATE_PIPE call to CICS. There must be a CICS region active with this APPLID on the same OS/390 host as the gateway or the EXCI initialization will fail and will display the message: `E_XCI010 II_EXCI_APPL_ID missing or invalid.  Unable to initialize EXCI.`	*IICDCICS*
NETNAME=	Connection NETNAME value to use for the INITIALIZE_USER call to CICS. This parameter must correspond to the NETNAME value specified on the CONNECTION definition of the EXCI pipe. An invalid NETNAME value will generate the following error message: `EXCI013 OPEN_PIPE failed: Reason = 609, Subreason-1 =104.` See the IBM document, *CICS/ESA: External CICS Interface (SC33-1390-00)* for a detailed explanation of the Reason and Subreason codes.	*BATCHCLI*

IGWFBLD Statement: Specifying IIVP Sysgen Parameters

The IGWFBLD statement specifies global default IIVP parameters and builds the IIVP stage2 jobstream.

Note: This statement must be last in the stage1 input.

The IGWFBLD statement is defined as follows:

```
IGWFBLD TYPE=GEN,                                 GENERATE JOBSTREAM          X
        RELEASE='EC IDMS 2.3/0112 (MVS.390/00)', RELEASE ID                  X
        CICSLIB=,                                 CICS EXCI LOAD LIBRARY      X
        CICSMAC=,                                 CICS MACRO LIBRARY          X
        LIST=NO,                                  DON'T SHOW LISTING          X
        LINKLIB=,                                 DEFAULT LINKLIB             X
        VTAMLIB=,                                 DEFAULT VTAMLIB             X
        PROCLIB=,                                 DEFAULT PROCLIB             X
        VTAMLST=,                                 DEFAULT VTAMLST             X
        SORTLIB=,                                 DEFAULT SORTLIB             X
        EDBC='EDBC.V2R3',                         EDBC DATASET PREFIX         X
        UNIT=3380,                                EDBC DEFAULT UNITNAME       X
        VOLSER=MVSVOL,                            EDBC DEFAULT VOLSER         X
        PRODUCTS=(IDMSGW,EDBC),                   SPECIFY PRODUCTS            X
        SYSDA='SYSDA',                            DEFAULT WORK UNIT           X
        VTAMMAC='SYS1.MACLIB',                    INPUT AMODGEN MACLIB        X
        AMODGEN='SYS1.AMODGEN'                    INPUT AMODGEN MACLIB        X
        ASMBLR='ASMA90',                          IBM ASSEMBLER               X
        STORCLAS=                                 SMS  STORAGECLASS
```

Customize the parameters for the IGWFBLD statement as shown in the following table:

Parameter	Usage	Default
TYPE=	Type of IIVP sysgen: ■ *GEN* The IIVP stage1 should produce the stage2 jobstream if the return code is no higher than 4. If the return code is higher than 4, some default was taken that requires review. ■ *NOGEN* The IIVP stage1 should not produce the stage2 jobstream.	GEN
RELEASE=	Release level of the gateway installation. **Note:** This value must be entered as defined in the Release Notes with the product tape.	None
CICSLIB=	Supplies the name of the CICS EXCI load library. This library will be included in the EDBC server STEPLIB concatenation.	CICS.VXXX.SDFHEXCI

Parameter	Usage	Default
CICSMAC=	Supplies the name of the CICS macro library used to assemble EXCI interface procedures.	CICS.VXXX.SDFHMAC
LIST=	Specifies whether the output of the IIVP stage1 should be a summary or a detail listing: ■ NO Specifies only a summary listing, describing each parameter and its default. ■ YES Specifies an assembler listing showing how each statement expanded to produce the stage2 jobstream. Used for diagnosing errors in the IIVP stage1 generation process.	NO
LINKLIB=	Specifies the name of a data set in the installation's LNKLSTxx member of SYS1.PARMLIB.	EDBC.V2R3. BACK.LOAD
VTAMLIB=	SNA LU0 or SNA LU62 only: Name of the VTAM link library that is the target for the mode table for the application used by the SNA protocol servers.	EDBC.V2R3. NET.LOAD
PROCLIB=	Name of the OS/390 PROCLIB that is updated with cataloged procedures that are specific to the gateway.	EDBC.V2R3. FILES.PROCLIB
VTAMLST=	Name of the VTAMLST data set that is updated with the VTAM application node for the designated protocol server.	EDBC.V2R3. SAMPLE.CNTL
SORTLIB=	Name of SORTLIB used by the stage2 sysgen.	SYS1.SORTLIB
PREFIX=	Dataset prefix for the gateway installation data sets. You must create an OS/390 alias for the high-level index of the data sets if one does not already exist. The value for this parameter must match the #PREFIX value as defined in the stage0 job.	EDBC.V2R3
UNIT=	Name of the physical device where the installation data sets will be allocated.	3380
VOLSER=	Serial number of the volume on which the installation data sets will be allocated.	MVSVOL

Parameter	Usage	Default
PRODUCTS=	Products that are to be customized for installation, in parentheses, separated by a comma. **Note:** The valid values are IDMSGW and EDBC, which must be defined.	(IDMSGW,EDBC)
SYSDA=	OS/390 unit name used for workspace allocation.	SYSDA
VTAMMAC=	Name of VTAM macro library.	SYS1.MACLIB
AMODGEN=	Name of system macro library.	SYS1.AMODGEN
ASMBLR=	Specifies which IBM ASSEMBLER is to be used by the stage2 SYSGEN. Valid values are IEV90 and ASMA90.	ASMA90
STORCLAS=	Specifies the SMS storage class to use for the allocation of the Name Server files. This value will override the UNIT= and VOLSER= specifications.	UNIT= VOLSER=

The Stage1 Jobstream

The stage1 jobstream uses the stage1 input, EDBC.V2R3.FILES.ASM and EDBC.V2R3.FILES.MACLIB to produce the stage2 jobstream, a report, and a listing.

To do this, you need to customize the stage1 jobstream in the data set with the default name EDBC.V2R3.SAMPLE.CNTL(IGWFSTGS). The following section details this procedure.

Customizing and Executing the Stage1 Jobstream

The following procedure customizes and executes the stage1 jobstream:

1. Use the ISPF editor to edit the member IGWFSTGS in data set EDBC.V2R3.SAMPLE.CNTL.
   - Add an installation job statement.

- On the STAGE1 EXEC statement, customize the PREFIX parameter by changing the #PREFIX string to the value of the gateway installation data set prefix. If the value chosen is *EDBC.V2R3*, then use the following ISPF editor command to customize this value:

  ```
  C '#PREFIX' 'EDBC.V2R3' ALL
  ```

The name of the library AMODGEN may be different, depending on the version of OS/390 you are running.

2. Submit the job for execution.

   The job gets a return code of 0 or 4. A return code of 4 will occur if you specify ENABLE=NO for any protocol server.

   This job produces the following:

   - *The IIVP jobstream.*
     It contains the following jobs, which are placed in data set *EDBC.V2R3*.STAGE2.CNTL.

     IGWFVPA0
     IGWFVPS0*
     IGWFVPS1*
     IGWFVPS2
     IGWFVPS3
     IGWFVPS4**
     IGWFVPI0
     IGWFVPP0***
     IGWFVPN0
     IGWFVPZ9

     *Jobs IGWFVPS0 and IGWFVPS1 are not generated if INSTALL=UPGRADE is specified in the IGWFIDMS macro.

     ** Job IGWFVPS4 will be generated only when the cross memory interface is being used (that is, IGWFPIPE is not specified)

     *** Job IGWFVPP0 will be generated only when the LU62 interface is being used (that is, IGWFPIPE is specified)

   - *SYSPRINT Stage1 Report*.
     This file contains a summary of all parameters and defaults selected. It shows the products that have been selected for installation by the IIVP jobstream and lists the jobs that were created in the IIVP jobstream.

   - *SYSLIN IIVP Jobstream Listing*.
     This file contains a listing of the IIVP jobstream. This listing is provided for diagnostic and documentation purposes.

Submitting the Stage2 Jobstream Jobs

This section summarizes the procedure you must use to edit and release each of the jobs in the IIVP stage2 jobstream. The sections that follow these steps describe each job in detail.

1. Edit the jobs as required, following site standards to add jobcards, passwords, and user IDs.
2. Submit the jobs one at a time in the exact sequence specified below:

–	*IGWFVPA0*	*Job:*	Customizes logical symbols jobstream
–	*IGWFVPS0**	*Job:*	Allocates CA-IDMS gateway databases
–	*IGWFVPS1**	*Job:*	Creates the gateway system catalogs
–	*IGWFVPS2*	*Job:*	Populates the gateway system catalogs
–	*IGWFVPS3*	*Job:*	Builds sample database JCL
–	*IGWFVPS4***	*Job:*	Installs cross memory service programs
–	*IGWFVPI0*	*Job:*	Customizes server parameters
–	*IGWFVPN0*	*Job:*	Creates name server files
–	*IGWFVPP0****	*Job:*	Customizes LU62 interface
–	*IGWFVPZ9*	*Job:*	Starts the EDBC server

   * Jobs IGWFVPS0 and IGWFVPS1 are not generated if INSTALL=UPGRADE is specified in the IGWFIDMS macro.

   **Job IGWFVPS4 is generated when the cross memory interface is used.

   ***Job IGWFVPP0 is generated when the LU62 interface is used.

3. Check the completion status of each job after it is released.

   Each job must complete before the next one begins.

4. If a job produces the expected return codes, release the next job.
5. If you encounter an error, make the necessary corrections and resubmit the failed job.

Note: You can restart each job in the IIVP stage2 jobstream on a job boundary. If you need to return to the stage1 step, you can do so without any harm. The stage2 jobs delete any data sets that were previously created.

Functions of Stage2 Jobs

The following sections describe each job in the IIVP stage2 jobstream.

IGWFVPA0 Job: Customizing the Logical Symbols Jobstream

This job customizes the installation and run-time parameters that are called logical symbols. The following table lists the steps executed by the following IGWFPA0 job:

Stepname	Description	RETCODE
STEP010	Adds members to IIPARM PDS.	0
STEPS1nn	Adds subsystem entries to USERS.DATA file.	0 or 4
STEPU1nn	Adds user entries to USERS.DATA file.	0 or 4

The following table describes the libraries updated by IGWFVPA0 Job:

Datasets Updated	Description of Dataset
EDBC.V2R3.FILES.IIPARM	Logical symbol file.
EDBC.V2R3.FILES.USERS.DATA	EDBC authorization file.

IGWFVPS0 Job: Allocating CA-IDMS Gateway Database Files

This job uses IDCAMS and IEFBR14 to allocate all required database files. STEP010 is optional and is coded as IEFBR14. To delete existing files change IEFBR14 to IDCAMS.

STEP020 allocates the three files required for the gateway Secondary SQL Catalog. Step030 allocates files for the gateway and EDBCDBA areas.

Job IGWFVPS0 is not generated if INSTALL=UPGRADE is specified in the IGWFIDMS statement.

The following table describes the steps executed by IGWFVPS0 Job:

Stepname	Description	RETCODE
STEP010	IDCAMS delete files.	00
STEP020	IEFBR14 allocate SQL catalog.	00
STEP030	IEFBR14 allocate database files.	00

IGWFVPS1 Job: Creating CA-IDMS Gateway Databases

This job uses IDMSBCF, MVS IEWL, and IDMSLOOK to define and initialize all required gateway databases.

Steps 10-40 are optional and create the Secondary SQL Catalog called EDCSQL. This catalog (IDMS Areas DDLCAT, DDLCATX and DDLCATLOD) can be used instead of a user Secondary SQL Catalog. This catalog also contains the definitions of the gateway system catalogs and all gateway user and schema specifications.

Steps 050 through 080 define and create the gateway and EDBCDBA Segments, Files, and Areas. The gateway segment contains one index and one data area for the gateway Standard Catalogs (tables) and one index and one data area for the extended catalogs.

The EDBCDBA segment consists of an index and one data area. By default, gateway users will store their tables and other objects in this default database.

Job IGWFVPS1 is not generated if INSTALL=UPGRADE is specified in the IGWFIDMS macro.

Note: This job will run in central mode when modifying the DMCL and DBNAME table and local mode when formatting database areas.

The following table describes the steps executed by IGWFVPS1 Job:

Stepname	Description	RETCODE
STEP010*	Creates EDCSQL catalog.	00 or 04
STEP020*	Modifies DBNAME table.	00 or 04
STEP030*	Links DBNAME table.	00
STEP040*	Modifies CV DMCL.	00 or 04
STEP050	Creates EDBC/EDBCDBA segments.	00 or 04
STEP060	Modifies CV DMCL.	00 or 04
STEP070	Links CV DMCL.	00
STEP080	Runs IDMSLOOK on NEW DMCL.	00
STEP090	Copies DMCL and DBNAME table to CDMSLIB.	00
STEP100	Initializes (formats) EDCSQL files.	00
STEP110	Initializes (formats) EDCCAT and EDCDBA files.	00

*These steps are not generated if SECONDARY_CATALOG = OLD is specified in the IGWFIDMS macro.

IGWFVPS2 Job: Populating the Gateway

This job completes the definition of the optional EDCSQL Secondary SQL Catalog, creates the Standard/Extended Catalog definitions, populates the Standard Catalogs and creates the EDBCDBA default SCHEMA.

Note: This job should run in central mode.

The following table describes the steps executed by IGWFVPS2 Job:

Stepname	Description	RETCODE
STEP010*	Defines EDCSQL using TABLEDDL.	00
STEP020*	Defines EDCSQL using VIEWDDL.	00
STEP025	Drops Standard Catalogs.	00 or 08
STEP030	Creates Standard Catalog.	00 or 04
STEP035	Drops Standard Views.	00 or 08
STEP040	Creates Standard Views.	00
STEP050*	Creates ExtendedCatalog.	00

Stepname	Description	RETCODE
STEP060*	Creates Default SCHEMA.	00 or 04
STEP070	Updates EDCSQL statistics.	00

* These steps are not generated if INSTALL=UPGRADE is specified in the IGWFIDMS macro.

IGWFVPS3 Job: Building the IIVP User Database

This job creates and loads the IIVP sample database to use in verifying the installation. Before this job is run, the CV must be recycled or a 'vary new copy' must be done to refresh the new DMCL and Database Name Table created by the S1 job.

If the return code is 8, check to see if the object being dropped does not exist. If the object does not exist, a return code of 8 is correct.
The following table describes the steps executed by IGWFVPS3 Job:

Stepname	Description	RETCODE
STEP010	Customizes IIVPBIDM SQL script.	
STEP020	Copies IIVPIDM script to Files.SQL.	
STEP030	Loads and checks out sample database.	0 or 8

The EDBC Terminal Monitor uses scripts in the EDBC.V2R3.FILES.SQL data set to create the sample database. These scripts cannot be used with IDMS's IDMSBCF. Cycle the CV to incorporate the new DMCL and DBNAME table load modules for this job. Update the JCL that brings up the CV to include all libraries.

IGWFVPS4 Job: Install Cross Memory Service Programs

This job installs the CA-IDMS cross memory transaction programs into the library specified by the CMDSLIB=parameter of the IGWFIDMS macro. It also generates ADD PROGRAM and TASK sysgen statements for these programs.

This job is not generated if the IGWFPIPE statement is included in the stage1 input stream.

The following table describes the steps executed by this job:

Stepname	Description	Retcode
STEP010	LKED transaction programs	0 or 4
STEP020	Store LKED JCL into SAMPLE.CNTL	0
STEP030	Generate ADD PROGRAM/TASK sysgen statements and store into SAMPLE.CNTL	0

IGWFVPI0 Job: Customizing EDBC Parameters

This job customizes EDBC parameters. If system libraries are not the target of the steps in this process, you must copy the resulting files to the appropriate libraries before you can initialize EDBC.

The following table describes the steps executed by IGWFVPI0 Job:

Stepname	Description	RETCODE
STEP010	Assembles the Network Load Module.	0
STEP015	Linkedits the Network Load Module.	0
STEP017	Creates the ASMPSERV JCL Member.	0
STEP020	Creates APPL definition for LU0.	0
STEP030	Creates procedures for EDBC.	0
STEP040	Assembles Mode Table.	0 or 8
STEP050	Linkedits Mode Table.	0 or 8
STEP060	Creates EDBC startup procedure.	0
STEP070	Creates batch JCL for the EDBC server.	0
STEP080	Customizes installation JCL.	0
STEP090	Customizes IIPARM Clist.	0
STEP100	Customizes database procedure JCL.	0

You may get a return code of 8 for STEP040 and STEP050 if there are no SNA protocols being generated.

The JCL created in STEP060 is identical with the JCL created in STEP030. The copy is placed unconditionally in the EDBC proclib. If you want, you can place the copy created in STEP030 into an system proclib as well.

The following libraries are updated by theIGWFVPI0 Job:

- EDBC.V2R3.NET.LOAD
- EDBC.V2R3.SAMPLE.CNTL
- EDBC.V2R3.FILES.PROCLIB

IGWFVPN0 Job: Creating Name Server Files

The IGWFVPN0 job creates the data sets required by the name server component of the OS/390 EDBC server.

The following table describes the steps executed by IGWFVPN0 Job:

Stepname	Description	RETCODE
STEP010	Deletes existing name server data sets.	0
STEP020	Allocates the name server data sets as follows: Dataset Size -- *prefix*.FILES.NAME.IICOMSRV.*id* 1 Trk *prefix*.FILES.NAME.IIINGRES.*id* 1 Trk *prefix*.FILES.NAME.IILOGIN.*id* 1 Trk *prefix*.FILES.NAME.IINODE.*id* 1 Trk *prefix*.FILES.NAME.IIDB2.*id* 1 Trk *prefix*.FILES.NAME.IIDCOM.*id* 1 Trk *prefix*.FILES.NAME.IIIDMS.*id* 1 Trk *prefix*.FILES.NAME.IIIMS.*id* 1 Trk *prefix*.FILES.NAME.IIVSAM.*id* 1 Trk where: *prefix* is value of PREFIX= parameter id is the two character installation code for the gateway installation	0
STEP030	Initializes prefix.FILES.NAME.IIDB2.id.	0
STEP040	Initializes prefix.FILES.NAME.IIIMS.id.	0
STEP050	Initializes prefix.FILES.NAME.IIVSAM.id.	0
STEP060	Initializes prefix.FILES.NAME.IIDCOM.id.	0
STEP070	Initializes prefix.FILES.NAME.IIIDM.id.	0
STEP080	Adds IDMS to the server class file.	0

IGWFVPP0 Job: Customizing the CA-IDMS LU62 PIPE

This job installs the CA-DMS LU62 PIPE. It updates the VTAMLST and VTAMLIB libraries and generates CA-IDMS sysgen statements. The installer should consult with network and IDMS systems personnel before running this job.

The following table describes the steps executed by IGWFVPP0 Job:

Stepname	Description	RETCODE
STEP010	Assembles module IIPIPE.	0
STEP020	Linkedits module IIPIPE.	0
STEP030	Creates the ASMPIPE JCL member.	0
STEP040	Creates PIPE APPL definitions.	0
STEP050	Creates PIPE MODEENT definitions.	0
STEP060	Assembles PIPE MODEENT definitions.	0
STEP070	Linkedits PIPE LOGMODE entry.	0
STEP080	Creates PIPE ASMLOGM JCL member.	0
STEP090	Creates CA-IDMS IDMScvnum member.	0
STEP100	Linkedits CA-IDMS taskid transaction program.	4
STEP110	Creates linkedit CA-IDMS taskid transaction program JCL.	0

The following table describes the libraries updated by IGWFVPP0 Job:

Datasets Updated	Comments
Library specified by IGWFBLD VTAMLIB= or EDBC.V2R3.SAMPLE.CNTL	STEP020
Library specified by IGWFBLD VTAMLST= or EDBC.V2R3.NET.LOAD	STEP070
Library specified by EDBC.V2R3.SAMPLE.CNTL	STEP090 STEP110

IGWFVPZ9 JOB: Starting the EDBC Server

The IGWFVPZ9 job starts the EDBC server as a batch job.

The EDBC.V2R3.BACK.LOAD and EDBC.V2R3.NET.LOAD libraries must be APF authorized before this job is submitted for execution. If they are not authorized, the server will be terminated with an S047 abend.

Stepname	Description	RETCODE
EDBCSVR	Starts the EDBC server	0

Final Installation Procedures

This section outlines the remaining installation procedures.

Completing the Installation

The next two sections describe how to update members of SYS1.PARMLIB.

Defining the Gateway Subsystem to OS/390

To define the EDBC subsystem to OS/390, add an entry like the following to the IEFSSNxx member of SYS1.PARMLIB:

```
EDBC     FOR EDBC SERVER
```

You can change the name of the EDBC subsystem from the default value, EDBC. The value you specify for the subsystem in IEFSSNxx must match the values you specified for the INSTALL= parameter of the IGWFPSVR macro, and the ID= parameter of the IGWFINET macro.

APF Authorizing the Gateway Load Libraries

To allow EDBC to initialize successfully, you must add the following data sets to the IEAAPFxx member in SYS1.PARMLIB. These data sets have the default names and parameters shown below:

*EDBC.V2R3.***BACK.LOAD** *volser*,

*EDBC.V2R3.***NET.LOAD** *volser*,

The value for the volser must match the same value specified on the VOLSER parameter in the IGWFBLD macro.

Note: These libraries may be APF authorized temporarily using the OS/390 SETPROG APF operator command. Temporarily authorizing these libraries permits the gateway to be started without the need for an IPL. At the first opportunity, however, these libraries should be permanently authorized by IPLing the system using the updated IEFSSNxx and IEAAPFxx members.

Customizing the TSO Logon Procedure

You must allocate the data sets that are required to execute the CA-IDMS gateway from your TSO session. By modifying your TSO logon procedure, you can access the gateway from your TSO session.

1. To customize your TSO logon procedure, add the following DD statements to the TSO logon procedure, or place them in the DD concatenation sequence if the DD statement already exists. The data sets are shown here with the default names.

   ```
   //STEPLIB  DD  DSN=EDBC.V2R3.FRONT.LOAD,DISP=SHR
   //    DD  DSN=EDBC.V2R3.BACK.LOAD,DISP=SHR
   //    DD  DSN=EDBC.V2R3.NET.LOAD,DISP=SHR
    .
   ... see the following Note
    .
   //SYSPROC  DD  DSN=EDBC.V2R3.FILES.CLIST,DISP=SHR
   //SYSCTL   DD  DISP=SHR,DSN=idms.sysctl
   ```

2. Copy the IIPARM Clist into a location where TSO can access the SYSPROC concatenation.

Note: CA-IDMS load libraries must be added to the STEPLIB concatenation if they are not in the LINKLIST.

Verifying the Installation Functionality

You can now test access to the sample CA-IDMS database that was created during the installation procedure. Verify local access to this sample database using the Terminal Monitor, which runs in a TSO address space. The CA-IDMS gateway also runs under TSO when accessed locally.

1. Log on to TSO as the gateway DBA using the revised logon procedure created above.
2. In ISPF, select option **6** from the Primary Option Menu to enter TSO commands.

3. Issue a command such as the following to define TSO access to the CA-IDMS gateway. This example uses the defaults for the server subsystem, EDBC, and for the data set prefix, EDBC.V2R3.

   iiparm isvr(*edbc***) prefix(***EDBC.V2R3***) sabe(***idms***)**

   then press Enter.

4. When the Clist ends successfully, issue the following command to invoke the Terminal Monitor and connect to the sample database. This example uses the default value for the CA-IDMS subsystem, IDMS. The final parameter is the server class, which must be IDMS.

   sql idms/idms

   then press Enter.

5. To see the tables in the sample database, type the following command.

   help \g

6. To execute the sample queries for the sample database, type the following:

   \i '*EDBC.V2R3***.files.sql(iivpqidm)'**

7. To quit the Terminal Monitor, type:

   \q

Chapter

6

Configuring the EDBC Server

This chapter describes the procedures to:

- Install and configure communication interfaces
- Enable and test security interfaces

This chapter also describes EDBC configuration options:

- Force Inactive Timeout
- Local Time Zone
- Alternate Translation Tables

Install and Configure Communication Interfaces

EDBC supports the SNA_LU0, SNA_LU62, IBM TCP/IP, SNS TCP/IP, KNET TCP/IP, and CCI network protocols. The EDBC network configuration is defined when EDBC is installed by specifications in the stage1 IGWFPSRV statement.

Modifications to the network do not require that EDBC be reinstalled. Additions or changes can be made by updating and submitting job ASMPSERV in the SAMPLE.CNTL library.

EDBC network configuration is maintained as a load module with a default name of IIPSERV. An installation can maintain multiple network configurations. The network configuration used by EDBC is controlled by logical symbol II_NET_LMOD. The default is:

```
II_NET_LMOD =     IIPSERV;
```

SNA LU0 for OS/390

EDBC supports the IBM SNA LU0 protocol on OS/390. This can be used for OS/390-to-OS/390 access as well as access from other platforms to OS/390.

Requirements

The gateway is certified for IBM OS/390 SNA LU0 with the following configuration: ACF/VTAM version 3.1.1 or higher.

Installation and Configuration

To install SNA LU0 support for the EDBC server, code the following parameters on the IGWFPSVR statement:

```
TYPE=SNA_LU0
ACB=
NODENAME=
DLOGMODE=
MODETAB=
ENABLE=
```

VTAM ACB and LOGMODE definitions for the SNA_LU0 logical unit are generated and placed in the SAMPLE.CNTL library as members RTIAPPL and RTIMODE. These definitions should be given to personnel responsible for VTAM. The SNA_LU0 interface cannot be started until the VTAMLST and VTAMLIB libraries have been updated with the generated ACB and LOGMODE definitions and the newly defined ACB activated.

Starting and Stopping the SNA_LU0 Interface

The following logical in the ISVREDBC member of EDBC.V2R3.FILES.IIPARM determines whether the SNA_LU0 network interface is to be automatically started during server initialization:

```
II_PROTOCOL_SNA_LU0  =  YES;
```

The ENABLE= parameter of the stage1 IGWFPSVR statement determines the value of this logical.

The alternative to starting the interface during server initialization is to activate it dynamically by issuing the following Modify operator command:

```
F EDBC,ACT,PROT=SNA_LU0
```

To stop the interface, issue the following Modify operator command:

```
F EDBC,INACT,PROT=SNA_LU0
```

Note: The above command terminates active SNA_LU0 connections.

Connecting from a Remote Client

To connect to the EDBC server using the SNA_LU0 interface, a vnode entry must be created on the EDBC client. See the "Managing Network Communications" chapter in *EDBC Getting Started* for information on how to create vnode entries for SNA_LU0.

SNA LU0 Abend Codes and Messages

Various internal error conditions can cause the SNA LU0 interface to terminate abnormally (abend). When an abend occurs, the EDBC server continues functioning; however, the SNA LU0 interface stops functioning. The abend code and summary information generally appear on the OS/390 console and the server JES log. The active load module displayed in the summary information is IILU0PS. The following table lists the possible abend codes:

Abend	GTF Error Message	Program
1001	ERROR WHEN SETTING UP RPL	PS0PSIN
1002	NO MATCHING PCB ADDR FOUND	Session Init
1003	NO CONNECT PWQE FOUND FOR THIS PCB	
1004	OPNDST ACCEPT MACRO ERROR	
1005	CLSDST MACRO ERROR	
1006	INVALID USER DATA – SESSION REJECTED	
1001	ERROR WHEN SETTING UP RPL	PS0RCVA
1002	ERROR WHEN REISSUING RECEIVE ANY	Receive Any
1003	RECEIVED A NEGATIVE RESPONSE	
1004	NO PCB FOUND FOR RCVD MSG WITH SAME SESSION ID	
1005	ERROR WHEN CHECKING RPL FROM RCV ANY	
1006	ERROR WHEN BUILDING PWQE	
1008	ERROR WHEN CHECKING RPL FOR NEG RESPONSE	
1009	ERROR WHEN CHECKING RPL FROM SENDING NEG RESP	
1010	RU RECEIVED LARGER THAN MAX BUFFER SIZE	
1011	BUFFER PUT ERROR – PROBABLE BUCB OVERLAY	
1001	ERROR WHEN GETTING DATA FROM BUFFER	PS0RECV Receive Req
1002	ERROR WHEN SETTING UP RPL	PS0SEND
1002	ERROR WHEN ISSUING SEND	Send Request

SNA LU62 for OS/390

The EDBC supports for the IBM SNA LU62 protocol on OS/390. This can be used for OS/390-to-OS/390 access as well as access from other platforms to OS/390.

Requirements

The gateway has been certified for IBM OS/390 SNA LU62 support with the following configuration: ACF/VTAM version 3.2 or higher.

Installation and Configuration

To install SNA LU62 support for the EDBC server, code the following parameters on the stage1 IGWFPSVR statement:

```
TYPE=SNA_LU62
ACB=
NODENAME=
DLOGMODE=
MODETAB=
ENABLE=
```

VTAM ACB and LOGMODE definitions for the SNA_LU62 logical unit are generated and placed in the SAMPLE.CNTL library as members RTIAPPL and RTIMODE. These definitions should be given to personnel responsible for VTAM. The SNA_LU0 interface cannot be started until the VTAMLST and VTAMLIB libraries have been updated with the generated ACB and LOGMODE definitions and the newly defined ACB activated.

Starting and Stopping the SNA_LU62 Interface

The following logical in the ISVREDBC member of EDBC.V2R3.FILES.IIPARM determines whether the SNA_LU62 network interface is to be automatically started during server initialization:

```
II_PROTOCOL_SNA_LU62  =  YES;
```

The ENABLE= parameter of the stage1 IGWFPSVR statement determines the value of this logical.

The alternative to starting the interface during server initialization is to activate it dynamically by issuing the following Modify operator command:

```
F EDBC,ACT,PROT=SNA_LU62
```

To stop the interface, issue the following Modify operator command:

F EDBC,INACT,PROT=SNA_LU62

Note: The above command terminates active SNA_LU62 connections.

Connecting from a Remote Client

To connect to the EDBC server using the SNA_LU62 interface, a vnode entry must be created on the EDBC client. See the "Managing Network Communications" chapter in *EDBC Getting Started* for information on how to create vnode entries for SNA_LU62.

SNA LU6.2 VTAM Logmode Entries

Two types of SNA LU6.2 LOGMODE entries can be created. They are as follows:

- Independent Logical Unit

  This type of logical unit supports parallel sessions. Multiple SNA LU6.2 sessions are multiplexed over a single logical unit.

  The following SNA mode table entry is required to support an independent logical unit:

```
MODEENT LOGMODE=EDCLU62,  MODE TABLE NAME                   X
    TYPE=0,                                                 X
    FMPROF=X'13',    NEGOTIABLE BIND                        X
    TSPROF=X'07',    REQUIRED FOR LU 6.2 SESSION            X
    PRIPROT=X'B0',   REQUIRED FOR LU 6.2 SESSION            X
    SECPROT=X'B0',                                          X
    COMPROT=X'D0B1',                                        X
    PSDNPAC=X'01',                                          X
    SSDNPAC=X'01',                                          X
    SRCVPAC=X'01',                                          X
    RUSIZES=X'8989', MAX RU SIZE=(8*2**9)=4096              X
    PSERVIC=X'060200000000000000000300'
```

- Dependent Logical Unit

  This type of logical unit supports single sessions. A single SNA LU6.2 session is supported over a single logical unit.

  The following SNA mode table entry is required to support a dependent logical unit:

```
MODEENT  LOGMODE=EDCLU62, MODE TABLE NAME                   X
    TYPE=0,                                                 X
    FMPROF=X'13',    NEGOTIABLE BIND                        X
    TSPROF=X'07',    REQUIRED FOR LU 6.2 SESSION            X
    PRIPROT=X'B0',   REQUIRED FOR LU 6.2 SESSION            X
    SECPROT=X'B0',                                          X
    COMPROT=X'D0B1',                                        X
    PSDNPAC=X'01',                                          X
    SSDNPAC=X'01',                                          X
    SRCVPAC=X'01',                                          X
```

```
RUSIZES=X'8989', MAX RU SIZE=(8*2**9)=4096             X
PSERVIC=X'060200000000000000000C00'
```

Check the vendor of the client API to determine which mode table entry should be used. The stage2 generates a sample mode table entry for SNA LU6.2 that assumes that the client supports parallel sessions. If this is not correct, modify the ASMRMODE member of SAMPL.CNTL to reflect the correct mode table type.

Sense Code 08120007 and Possible Loop in VTAM

If you use the SNA LU 6.2 interface with many concurrent users, problems may occur where VTAM rejects incoming BINDS with a sense code of 08120007. This is a VTAM configuration issue. Should this occur, check the NCP gen to ensure that sufficient resources have been allocated by VTAM to support the SNA LU6.2 requirements. You can allocate resources for specific independent logical units by coding the TESSCB parameter on the LU statements defining that particular logical unit. Or a pool of resources can be allocated to be used by any independent logical unit by coding the AUXADDR and ADDSESS parameters on the BUILD statement of the NCP gen. In either case, you should specify values for the parameters that at least equal the desired maximum concurrent users.

KNET TCP/IP for OS/390

The network support for Fibronics KNET OS/390 TCP/IP allows EDBC to communicate with other platforms across a TCP/IP network.

Requirements

The server is certified for KNET OS/390 TCP/IP support with the following configuration: KNET TCP/IP for OS/390 Release 1.3 (or higher).

EDBC uses the KNET API, which implements an SNA interface from the EDBC to the KNET address space. The KNET address space communicates directly with the Fibronics K200 or K2000 controller to provide connectivity to the workstations attached to an Ethernet LAN.

Installation and Configuration

To install TCP_KNET support for the EDBC server, code the following parameters on the stage1 IGWFPSVR statement:

```
TYPE=TCP_KNET
PLU=
PORT=
ENABLE=
```

Starting and Stopping the TCP_KNET Interface

The following logical in the ISVREDBC member of EDBC.V2R3.FILES.IIPARM determines whether the TCP_KNET network interface is to be automatically started during server initialization:

```
II_PROTOCOL_TCP_KNET  =  YES;
```

The ENABLE= parameter of the stage1 IGWFPSVR statement determines the value of this logical.

The alternative to starting the interface during server initialization is to activate it dynamically by issuing the following Modify operator command:

```
F EDBC,ACT,PROT=TCP_KNET
```

To stop the interface, issue the following Modify operator command:

```
F EDBC,INACT,PROT=TCP_KNET
```

Note: The above command terminates active TCP_KNET connections.

Connecting from a Remote Client

To connect to the EDBC server using the TCP_KNET interface, a vnode entry must be created on the EDBC client. See the "Managing Network Communications" chapter in *EDBC Getting Started* for information on how to create vnode entries for TCP_KNET.

IBM TCP/IP for OS/390

The network protocol support for IBM's OS/390 TCP/IP product enables EDBC to communicate with other platforms across a TCP/IP network.

Requirements

The gateway is certified for IBM OS/390 TCP/IP support with the following configuration:

- MVS/ESA, OS/390, or z/OS
- IBM TCP/IP for OS/390 V3R2 and above

Installation and Configuration

To install IBM TCP/IP support for the EDBC server, code the following parameters on the stage1 IGWFPSVR statement:

```
TYPE=TCP_IBM
PORT=
USERID=
ENABLE=
```

Starting and Stopping the TCP_IBM Interface

The following logical in the ISVREDBC member of EDBC.V2R3.FILES.IIPARM determines whether the TCP_IBM network interface is to be automatically started during server initialization:

```
II_PROTOCOL_TCP_IBM  =  YES;
```

The ENABLE= parameter of the stage1 IGWFPSVR statement determines the value of this logical.

The alternative to starting the interface during server initialization is to activate it dynamically by issuing the following Modify operator command:

```
F EDBC,ACT,PROT=TCP_IBM
```

To stop the interface, issue the following Modify operator command:

```
F EDBC,INACT,PROT=TCP_IBM
```

Note: The above command terminates active TCP_IBM connections.

Connecting from a Remote Client

To connect to the EDBC server using the TCP_IBM interface, a vnode entry must be created on the EDBC client. See the “Managing Network Communications” chapter in *EDBC Getting Started* for information on how to create vnode entries for TCP_IBM.

IBM TCP/IP Problem Diagnosis

The EDBC TCP/IP protocol server issues an error message whenever an IBM TCP/IP error occurs. The format of the error message is as follows:

```
subsysid: TCP-IBM "function" RETCODE = -00000001 ERRNO= error number
```

where:

subsysid:	= EDBC subsystem id
function	= TCP/IP operation (RECEIVE, SEND, LISTEN, and so on)
error number	= TCP/IP error return code

See the *IBM TCP/IP Application Programming Interface Reference* for a complete description of error return codes.

CCI for OS/390

EDBC supports the CCI protocol on OS/390. This is used for OS/390-to-OS/390 access.

Requirements

The gateway has been certified for OS/390 CCI support with the following configuration: CA90s level 9312 or higher.

Installation and Configuration

To install CCI support for the EDBC server, code the following parameters on the stage1 IGWFPSVR statement:

```
TYPE=CCI
PRODID=
ENABLE=
```

Starting and Stopping the CCI Interface

The following logical in the ISVREDBC member of EDBC.V2R3.FILES.IIPARM determines if the CCI network interface is to be automatically started during server initialization:

```
II_PROTOCOL_CCI  =  YES;
```

The ENABLE= parameter of the stage1 IGWFPSVR statement determines the value of this logical.

The alternative to starting the interface during server initialization is to activate it dynamically by issuing the following Modify operator command:

F EDBC,ACT,PROT=CCI

To stop the interface, issue the following Modify operator command:

F EDBC,INACT,PROT=CCI

Note: The above command terminates active CCI connections.

Connecting from a CCI Client

The utility, EDBCNETU, is used to configure a client. The node information for CCI is as follows:

```
Protocol:              CCI
Remote Node Address:   CCI     System ID of other node
Remote Listen Address  CCI     EDBC subsystem ID
```

The Remote Node Address specifies the System ID of the CCI system that you are connecting to. The Remote Listen Address is the Subsystem ID of the EDBC server.

CCI Abend Codes and Messages

Various internal error conditions can cause the CCI interface to abnormally terminate (abend). When an abend occurs, EDBC continues functioning; however, the CCI interface stops functioning and no new connections can be made until the CCI interface is restarted. The abend code and summary information usually appears on the OS/390 console and the gateway JES log. The active load module displayed in the summary information is IICCIPS. The following table lists the possible abend codes for the CCI protocol server:

Abend	GTF Error Message	Program
1110	RU RECEIVED LARGER THAN MAX BUFFER SIZE	PSCRCVA
1111	BUFFER PUT ERROR - PROBABLE BUCB OVERLAY	Receive any
1202	ERROR WHEN GETTING DATA FROM BUFFER	PSCDFSM receive req

SNS/TCP for OS/390

The network protocol support for SNS/TCP on OS/390 is functionally the same as IBM's OS/390 TCP/IP protocol support. That is, it allows EDBC to communicate with other platforms across a TCP/IP network.

Requirements

The gateway has been certified for SNS/TCP with the following configuration: SNS/TCP Version 2.0 or higher.

Installation and Configuration

To install SNS TCP/IP support for the EDBC server, code the following parameters on the stage1 IGWFPSVR statement:

```
TYPE=TCP_SNS
APPLID=
PORT=
SYSID=
ENABLE=
```

Starting and Stopping the TCP_SNS Interface

The following logical in the ISVREDBC member of EDBC.V2R3.FILES.IIPARM determines whether the TCP_SNS network interface is to be automatically started during server initialization:

```
II_PROTOCOL_TCP_SNS  =  YES;
```

The ENABLE= parameter of the stage1 IGWFPSVR statement determines the value of this logical.

The alternative to starting the interface during server initialization is to activate it dynamically by issuing the following Modify operator command:

```
F EDBC,ACT,PROT=TCP_SNS
```

To stop the interface, issue the following Modify operator command:

```
F EDBC,INACT,PROT=TCP_SNS
```

Note: The above command terminates active TCP_SNS connections.

Connecting from a Remote Client

To connect to the EDBC server using the TCP_SNS interface, a vnode entry must be created on the EDBC client. See the "Managing Network Communications" chapter in *EDBC Getting Started* for information on how to create vnode entries for TCP_SNS.

Enable and Test Security Interfaces

EDBC interfaces to IBM's Resource Access Control Facility (RACF) and the Computer Associates' Access Control Facility (CA-ACF2) and Top Secret Security (CA-TSS) security products. The security interface is activated by the following logical in the ISVREDBC member of EDBC.V2R3.FILES.IIPARM:

```
II_SECURITY =
```

The SECURITY = parameter of the stage1 IGWFINET statement determines the value of this logical. The options are RACF, ACF2, TSS or NONE.

The interface to the security products occurs in an exit routine that is invoked when a user attempts to connect to EDBC. The exit routine in turn calls the security product to authenticate the user. There is an exit routine for each of the supported security products.

The sources for these routines are distributed in the FILES.ASM library as members IIRACF, IIACF2, and IITSS.

IBM Resource Access Control Facility (RACF)

EDBC supports IBM's Resource Access Control Facility (RACF). This facility performs authentication checking and resource control for an OS/390 system. It requires that a remote client be authenticated by RACF before a connection is established.

When RACF security is requested, the EDBC server uses RACF to validate the user ID and password. This validation occurs during connection processing.

If the validation fails, then the connection request is rejected and RACF issues message ICH408I. If the validation of the user ID/password is successful, then RACF returns a pointer to a security control block.

The interface to RACF uses the IBM RACINIT interface to issue the logon validation requests from EDBC clients to RACF.

Installing and Customizing the RACF Interface

This section describes how to customize EDBC to enable RACF security. Complete the following steps:

1. Install the EDBC server.
2. Create a RACF profile for the EDBC server.
3. Create a RACF profile for each EDBC client.
4. Define the user created in the RACF profile to EDBC.
5. Create a vnode definition on the EDBC client.
6. Set the II_SECURITY logical symbol.
7. Test the RACF interface.

Creating a RACF Profile for the Server

Create a RACF identifier for the EDBC task.

Creating a RACF Profile for Each Gateway Client

Create a RACF profile for each user who will access the server. This profile should have the same characteristics as that of a TSO user.

Defining the User to EDBC

The user created in the RACF profile definition must be defined to the server. See the "Maintaining the Gateway" chapter for instructions on adding new users.

Creating a Vnode on the Client

On the client use the EDBC Network utility to create a vnode entry.

See *EDBC Getting Started* for instructions on creating vnodes.

Setting the II_SECURITY Logical Symbol for the EDBC Server

To enable the RACF security interface, add the following logical symbol to the ISVREDBC member in EDBC.V2R3.FILES.IIPARM.

```
II_SECURITY = RACF;
```

Testing the RACF Interface

Use the following procedure to test the RACF interface:

1. Start the EDBC server.
2. On the EDBC client, invoke the EDBC Network utility.
3. Right-click the newly created vnode and select SQL Test.
4. If the SQL test panel appears, the RACF interface has been successfully installed.

Computer Associates Access Control Facility 2

The server supports the Computer Associates Access Control Facility 2 (CA-ACF2). This facility performs authentication checking and resource control for the OS/390 system. It ensures that a remote client is authenticated before a connection is established. If the user ID/password fails, then the remote connection request is rejected. If the user ID/password is valid, the connection processing continues.

The interface to CA-ACF2 uses the IBM Security Access Facility (SAF) to issue the logon validation requests from the EDBC client to CA-ACF2.

Installing and Customizing the CA-ACF2 Interface

To enable the CA-ACF2 security interface, complete the following steps:

1. Create a CA-ACF2 Logon IDentifier (LID) for the server.
2. Create a CA-ACF2 protection record.
3. Create a CA-ACF2 LID for each user.
4. Define the user to the server.
5. Create a vnode for each EDBC client.
6. Set the II_SECURITY logical symbol for the server.
7. Test the CA-ACF2 interface.

Creating a Server CA-ACF2 LID

Create a CA-ACF2 identifier for the EDBC task. It should have the same characteristics as CICS. Verify that the following fields are set:

- MUSASS
- STC

Creating a CA-ACF2 LID for Each User

Create a CA-ACF2 LID for each user who will access the server. This LID should have the same characteristics as that of a TSO user.

Defining the User to EDBC

The user created in the RACF profile definition must be defined to the server. See the “Maintaining the Gateway” chapter for instructions on adding new users.

Creating a Vnode on the Client

On the client use the EDBC Network utility to create a vnode entry.

See *EDBC Getting Started* for instructions on creating vnodes.

Setting the II_SECURITY Logical Symbol for the EDBC Server

To enable the CA-ACF2 security interface, add the following logical symbol to the ISVREDBC member in the EDBC.V2R3.FILES.IIPARM.

```
II_SECURITY = ACF2;
```

Testing the CA-ACF2 Interface

Use the following procedure to test the CA-ACF2 interface:

1. Start the EDBC server.
2. On the EDBC client, invoke the EDBC Network utility.
3. Right-click the newly created vnode and select SQL Test.
4. If the SQL test panel appears, the RACF interface has been successfully installed.

Computer Associates Top Secret Security Facility

The server supports the Computer Associates Top Secret Security (CA-TSS). This facility performs authentication checking and resource control for an OS/390 system. It ensures that a remote client is authenticated before a connection is established. If the user ID/password fails, the remote connection request is rejected. If the user ID/password is valid, then the connection processing continues.

The interface to CA-TSS uses the IBM RACINIT to issue the logon validation requests from the server to CA-TSS.

Installing and Customizing the CA-TSS Interface

To enable CA-TSS security, complete the following steps:

1. Create a CA-TSS Facility for the server.
2. Create a CA-TSS Access Control Identifer (ACID) for the server.
3. Create a CA-TSS ACID for each user.
4. Define the user to the server.
5. Create a vnode on the EDBC client.
6. Set the II_SECURITY logical symbol for the EDBC server.
7. Test the CA-TSS interface.

Creating a Server CA-TSS Facility

A CA-TSS facility must be created to authorize the server to issue RACINT requests.

The following example shows the suggested control options for the server:

```
GATEWAY
INITPGM=IIG ID=D TYPE=31
ATTRIBUTES=ACTIVE,SHRPRF,NOASUBM,NOTENV,NOABEND,
ATTRIBUTES=MULTIUSER,NOXDEF,LUMSG,STMSG,SIGN(M),
ATTRIBUTES=NOPSEUDO,INSTDATA,NORNDPSW,AUTHINIT,
ATTRIBUTES=NOPROMPT,NOMENU,NOAUDIT,NORES,NOMRO,
ATTRIBUTES=WARNPW,NOTSOC,NOTRACE,NOLAB,NOEXTEND,
ATTRIBUTES=NODORMPW,NONPWR,NODATACOMXTND
MODE=FAIL
LOGGING=ACCESS,INIT,SMF,MSG
UIDACID=8 LOCKTIME=000 DEFACID=*NONE* KEY=8
```

Creating a Server CA-TSS ACID

Create a CA-TSS identifier for the EDBC task. This Access Control Identifier (ACID) should be associated with the OS/390 started task. In this example, the ACID name is EDBCI1. The ACID must be able to access the server CA-TSS facility and the CA-IDMS CA-TSS facility.

Creating a CA-TSS ACID for Each User

Create a CA-TSS ACID for each usesr who will access the server. This ACID should have the same characteristics as that of a TSO user.

The client must use TSO to set the correct password for this authorization ID. The gateway does not support the setting of the CA-TSS password from a gateway client.

Defining the User to EDBC

The user created in the CA-TSS profile definition must be defined to the server. See the “Maintaining the Gateway” chapter for instructions on adding new users.

Creating a Vnode on the Client

On the client use the EDBC Network utility to create a vnode entry.

See *EDBC Getting Started* for instructions on creating vnodes.

Setting the II_SECURITY Logical Symbol for the EDBC Server

To enable the CA-TSS security interface, add the following logical symbol to the ISVREDBC member in EDBC.V2R3.FILES.IIPARM.

```
II_SECURITY = TSS;
```

Testing the CA-TSS Interface

Before you test the CA-TSS interface, you should have successfully completed installing the gateway.

Use the following procedure to test the CA-TSS interface:

1. Start the EDBC server.
2. On the EDBC client, invoke the EDBC Network utility.
3. Right-click the newly created vnode and select SQL Test.
4. If the SQL test panel appears, the RACF interface has been successfully installed.

Force Inactivate Timeout

The force inactivate timeout facility cleans up threads that are hung as a result of circumstances outside the control of the server. The facility frees up user threads after a pre-specified period of time, following the failure of a disconnect or inactivate. It cannot be invoked directly by the user, but must first be preceded by the failure of a user disconnect, an operator inactivate, or an inactivate user timeout.

To activate the force inactivate timeout facility, set the following symbol in EDBC.V2R3.FILES.IIPARM(ISVREDBC).

```
II_FORCE_TMOUTINT = nn;
```

where *nn* is the number of minutes that elapses before a force inactivate is executed against the thread that did not successfully inactivate or disconnect. The default is five minutes.

If the parameter is not specified, or if it is set to 0, the inactivate user timeout facility is not activated and therefore, no force inactivates occur. Error messages normally accompany a force inactivate timeout.

Note: Do not confuse the force inactivate timeout logical with the SRV_TMOUTINT or SRV_TMOUTINT or II_INACTV_TMOUTINT logicals. SRV_TMOUTINT controls how often all of the timers (including the force inactivate timer) are checked. II_INACTV_TMOUTINT controls the length of time a user session can have no activity before it is automatically inactivated. For more information on these logicals, see the “Server Logical Symbols and the IIPARM Clist” appendix.

You may also set the II_FORCE_TMOUTINT symbol during stage2 of the IIVP.

Local Time Zone

The EDBC server logical II_TIMEZONE is used to specify the difference in hours between Greenwich mean time (GMT) and the local time zone where the EDBC server is running. Specify this logical in EDBC.V2R3.FILES.IIPARM(ISVREDBC) as follows:

```
II_TIMEZONE = 'n';
```

where *n* specifies the local time zone displacement. If this parameter is not specified, it defaults to 0.

When the EDBC server is initialized, the value of the II_TIMEZONE logical is compared to the system GMT time zone displacement. If you have defined the logical to match the value of your local time zone, you will see the following message at startup time:

```
EDBC: II_TIMEZONE set to n
```

If the value you set for II_TIMEZONE does not match the value of your local time zone as set in SYS1.PARMLIB(CLOCKxx), you will see the following message at startup time:

```
EDBC: II_TIMEZONE set to n
EDBC: II_TIMEZONE (n) does not match local timezone(n)
EDBC: II_TIMEZONE DATE conversion may be incorrect
```

If II_TIMEZONE is defined incorrectly on either the server or client installation, dates will appear incorrect.

The gateway does not support the II_DATE_FORMAT logical; therefore, all date input strings must be in US format. See the *EDBC OpenSQLReference Guide* for a list of valid US date formats.

Year 2000 Support

The installation accommodates the year 2000 by allowing you to set the II_DATE_CENTURY_BOUNDARY logical symbol. See the "Server Logical Symbols and the IIPARM Clist" appendix for more information about this logical symbol.

Alternate Translation Tables

The server can use alternate translation tables that allow the default table (Country Extended Code Page 037 for the USA) to be overridden.

Note: This capability only applies to single byte character sets.

To override the default table, use the IIPARM logical, II_GCC_TRANSLATE_TABLE, which specifies the name of the alternate translation table to be used. For example:

```
II_GCC_TRANSLATE_TABLE  = 'CECP277';
```

The above example would bring in the Denmark/Norway translate table. Some, but not all, alternate translation tables are supplied with the server. Members CECPnnn (where nnn is Extended Code Page number) are supplied in the FILES.ASM library in source form and the compiled forms are in the BACK.LOAD library.

A job (member CECPNNN in the SAMPLE.CNTL library) can be run, with modifications, to compile an alternate translation table.

The translate table causes the server to change incoming (GCC Network Standard single byte characters [ASCII with extensions]) characters into local EBCDIC characters. (A reverse table is built for outgoing characters.)

To build a new table, start with the USA table CECP037 and make changes where appropriate. The notes in the CECP037 table will assist you with the needed changes. In summary, there are four types of changes:

1. Code Page Name—CECPnnn where nnn is the CECP identifier.
2. Tilde substitute value—the hex value of whateverserver value x'7E' is translated to.
3. Dollar Sign substitute value—the hex value of whatever server value x'24' is translated to.
4. Server substitute values (multiple)—in the CECP037 source, each server value that translates into a non-zero value will indicate, in comments, what the server value represents. All of these should be reviewed and changes made as required.

When an OS/390 client connects to an EDBC server, the server must force the client to translate its outgoing messages. This is accomplished by including member FORCEHET in the IIPARM concatenation in the server JCL.

Chapter 7

Maintaining the Gateway

The gateway and the EDBC server run in a single address space. This address space may require periodic maintenance. This chapter assumes that one person, probably an OS/390 system programmer, is designated to act as the gateway DBA.

This chapter describes how the gateway DBA carries out the tasks:

- Managing CA-IDMS tables created or accessed through the gateway
- Bringing up and shutting down the EDBC server
- Carrying out a range of operations with the OS/390 modify command, such as activating or inactivating a specified protocol server, or turning tracing on or off for troubleshooting
- Enabling a new user to access CA-IDMS databases through the gateway
- Creating a second EDBC server on the same OS/390 host

Database Management Functions

The gateway interacts with CA-IDMS as a standard CA-IDMS application. All CA-IDMS data accessed through the gateway remains completely under the control of CA-IDMS and is governed by CA-IDMS rules. As with any other CA-IDMS application, CA-IDMS determines data access permissions, optimum data access path, and data integrity and recovery.

The DBA for each CA-IDMS Central Version should continue to create, destroy, back up, and recover CA-IDMS data using the functions provided by CA-IDMS. For further information on managing CA-IDMS Central Versions, consult your CA-IDMS documentation.

Starting and Stopping the EDBC Server

The procedures in the following sections start and stop the EDBC server.

Starting the EDBC Server as a Started Task

To start the EDBC server, enter the following OS/390 operator command:

S *procname.identifier*

For example:

S EDBCI1.EDBC

The following messages appear on the OS/390 console:

```
EDBC: EDBC  MVS Server Default Version for CPU id xx-xxxxxx-xxxx
EDBC: Initializing Operator Interface Subtask
EDBC: Initializing Housekeeping Subtask
EDBC: Initializing communication queue servers
EDBC: Initializing communication drivers
EDBC: SNA_LU0 Protocol Server Initialization Begun
EDBC: SNA_LU0 Opened ACB IIS1GWS1
EDBC: SNA_LU0 Protocol Server Initialization Complete
EDBC: TCP_IBM Protocol Server Initialization Begun
EDBC: TCP_IBM Listening on Port 2610
EDBC: TCP_IBM Protocol Server Initialization Complete
EDBC: TCP_SNS protocol server Initialization Begun
EDBC: IDMS connection ACTivated
EDBC: EDBC Server Initialization Complete
```

Note: This example shows two of the protocol servers initializing. You may have other protocol servers installed at your site.

After initialization is successfully completed, EDBC is ready to process requests.

Stopping the EDBC Server

Stopping the EDBC server will cause all active connections to be terminated.

To shut down EDBC, enter the following OS/390 operation command:

P *identifier*

For example, with the default installation value for the identifier, this command is:

P EDBC

After you issue this command, the following messages appear on the OS/390 console:

```
EDBC: EDBC Server shutdown in progress
EDBC: Wait for Operator Interface to end
EDBC: Operator Interface ended
EDBC: SNA_LU0 Protocol Server Termination Begun
EDBC: SNA_LU0 Protocol Server Termination Complete
EDBC: TCP_IBM Protocol Server Termination Begun
EDBC: TCP_IBM Protocol Server Termination Complete
EDBC: Housekeeping shutdown complete
EDBC: Terminating communication queue servers
EDBC: EDBC Server shutdown complete (rc=0)
```

Starting the EDBC Server as a Batch Job

To start up the EDBC server as a batch job, do the following:

1. Use either one of the following members from the SAMPLE.CNTL library to start the EDBC server as a batch job:

   – *xxxx*EDBC where *xxxx* is the OS/390 subsystem ID

   – START*yy* where yy is the server installation ID

2. Submit the job for execution.

   The console displays the series of messages shown in the previous section, terminating in the message:

   ```
   EDBC: EDBC server initialization complete
   ```

The Gateway Hot Connect Function

The gateway *hot connect function* opens a connection between a gateway and a DBMS. It reduces the time it takes to connect to a DBMS.

The hot connect opens tables in the DBMS address space and keeps them open for the life of the hot connection. Because these tables are already open, the path lengths of subsequent connections are shortened and thus connect time is reduced.

The hot connect task runs in the EDBC server address space and can be started during server initialization or by an OS/390 operator command. Startup during initialization is determined by the IDMS_HOT_CONNECT logical symbol of the ISVR member of IIPARM. Operator commands to ACTivate and INACTivate the hot connection task are described in the Maintaining the Gateway section.

Note: For CA-IDMS, the hot connect behavior differs depending on how the gateway interfaces to the CA-IDMS CV. When the LU62 interface is used, the behavior is as described above. With the cross memory interface, the hot connect is used to start and stop EDBC cross memory services. No communications can occur between the CA-IDMS gateway and a CA-IDMS CV until the cross memory services are started on both the EDBC server and CA-IDMS CV. CA-IDMS tasks EDBCSTRT and EDBCSTOP control the starting and stopping of cross memory services for a CA-IDMS CV.

Maintaining the Gateway

You can use the OS/390 modify command to carry out a series of operations for the gateway. From the OS/390 operator console you can:

- Display active users and internal service tasks, with information about the status of each thread
- Activate a protocol server or gateway hot connection
- Inactivate a protocol server, user ID, user context, or gateway hot connection
- Set tracing to debug a protocol server
- Turn off tracing
- Dump the entire address space for troubleshooting
- Get help for this utility

These commands are all variations of the following format:

F *identifier*, *command*

A blank or a comma is a valid delimiter. Extraneous blanks produce errors. The uppercase characters in the commands listed in the following table denote the acceptable abbreviations for the commands.

For example, to display a list of all active threads for the gateway, when the subsystem ID is EDBC, the command is:

F EDBC,D A

The following table summarizes this range of commands and the purposes they serve:

Command	Purpose	Parameters	Options
ACTivate	Activates a protocol server or LU62 PIPE and makes it available to accept incoming communications. Also can start a gateway hot connection. Example: **F EDBC,ACT,PROT=KNET** Example: **F EDBC,ACT,CONNECT=IDMS**	PROT=	CCI SNA_LU0 SNA_LU62 TCP_KNET TCP_IBM TCP_SNS
		PIPE=	IDMSLU62
		CONNECT=	IDMS
INACTivate	Cancels the specified thread, which can be a user ID, user context, protocol server, PIPE, or gateway hot connection. Example: **F EDBC,INACT,ID=ABC00** If a specified user ID is inactivated, the user is gracefully shutdown, with an appropriate screen prompt. All other users and tasks remain active. Example: **F EDBCINACT,CONTEXT=02d08c10** If a specified user context is inactivated, the user with that unique identifier is gracefully shut down. Other users with the same user ID, but different contexts, remain active. Example: **F EDBC,INACT,PROT=LU0** If one protocol server is shut down, all threads associated with it are also shut down. If another protocol server is installed, the latter remains active, along with the entire gateway. Example: **F EDBC,INACT,CONNECT=IDMS** This terminates the hot connection to the specified gateway. All other threads are unaffected.	ID=	User ID
		CONTEXT=	Context
		PROT=	CCI SNA_LU0 SNA_LU62 TCP_KNET TCP_IBM TCP_SNS
		PIPE=	IDMSLU62
		CONNECT=	IDMS

Command	Purpose	Parameters	Options
TRace	Turns on tracing to provide dynamic information about the activity of the specified protocol server or PIPE for debugging. Example: **F EDBC,TR,PROT=LU0** **Note:** Do not turn on tracing except when recommended by Computer Associates' Technical Support.	PROT= PIPE=	CCI SNA_LU0 SNA_LU62 TCP_KNET TCP_IBM TCP_SNS IDMSLU62
NOTRace	Turns off tracing for the specified protocol server or PIPE. Example: **F EDBC,NOTR,PROT=KNET**	PROT= PIPE=	CCI SNA_LU0 SNA_LU62 TCP_KNET TCP_IBM TCP_SNS IDMSLU62
DUMP	Dumps the entire address space for troubleshooting purposes. Example: **F EDBC,DUMP,ASID=14** **Note:** Give this command only when recommended by Computer Associates' Technical Support.	ASID=	Numeric identifier for OS/390 address space
HELP	Provides a brief display of all the options of the modify command for use by the OS/390 operator, with syntax and parameters. Example: **F EDBC,HELP**	None	None
Display Active	Provides information on all active threads in the gateway address space. **Note:** See the Display Active Command section for more detailed information on this command. Example: **F EDBC, D A**	None	None

Display Active Command

Enter the following command to display the active threads on your subsystem:

F EDBC, D A

This command presents a display similar to the following example on your screen:

```
IEDBC Display of Active Threads: 017
* Context  Protocol Service  User   I/O#   Inact   Status  *
|----------------------------------------------------------|
| 05280010 Local    Operator
| 05280410 Local    Hkeeping
| 05280810 Local    Namesvr
| 05280C10 Local    EDBC
| 05281010 Local    Hotmntr
| 05281410          HOTCONN   IDMS
| 05281810 Local    Monitor
| 05281C10 SNA_LU0  ProtSrvr
| 05282010 SNA_LU62 Protsrvr
| 05282410 TCP_IBM  ProtSrvr
| 05282810 TCP_SNS  Protsrvr
| 05282C10 CCI      ProtSrvr
| 05283C10 IDMSLU62 Pipe            119450    0   *ACTive*
| 05283010 TCP_SNS  IDMS     usro01     43    0   130.119.5.7..1261
| 05283410 CCI      Listener             0   11   *Listen*
| 05283810 TCP_SNS  Listener             0    0   *Listen*
| Threads (Max=64, Curr=15, User=1)
+ Timeout (Inactv=120, Force=5)
```

The following table describes the fields in this display:

Field	Description	Predefined Value
Context	Unique 8-character identifier used to identify a gateway thread. Since more than one user can share a user ID, the context gives you a way to identify an individual user. Each internal thread in the gateway is assigned a unique token. Use the token to cancel a thread via the following operator command: **F EDBC,INACT,CONTEXT=**	None

Field	Description	Predefined Value
Protocol	Protocol associated with the thread.	■ Local - Internal Thread ■ CCI ■ SNA_LU0- SNA LU0 protocol ■ SNA_LU62 - SNA LU6.2 protocol ■ TCP_IBM - IBM TCP/IP protocol ■ TCP_KNET - KNET TCP/IP protocol ■ TCP_SNS - Interlink SNS/TCP protocol
Service	Descriptive name of each gateway thread. There is one *uuuuuu* entry for each remote connection.	■ Operator - Operator thread ■ Hkeeping - Utility thread ■ Namesvr - Name server thread ■ EDBC - Initialization thread ■ *Listen*- Listener thread ■ *uuuuuu* - Name of remote user ID ■ IDMS LU62 – CA-IDMS LU62 PIPE ■ HOTMNTR - Hot connection monitor ■ HOTCONN - Gateway hot connection
User	Server class of a gateway thread. This field is valid only for gateway threads.	■ CA-IDMS - CA-IDMS gateway thread
I/O #	Total number of sends and receives processed by the gateway from the beginning of the session. Shows the level of gateway activity. This field is valid only for gateway threads. The range of values is 0 to 9999999.	None
Inact	Length of time, in minutes, since the last I/O. This field can be used to determine which threads are idle. Indicates how long a *Listen* thread has been outstanding for the protocol server.	None

Field	Description	Predefined Value
Status	Provides current STATUS of the thread.	■ IP address and port number ■ *LISTEN* ■ TERM_PND ■ NO_PROT_SRVR ■ PCB_NOT_FOUND ■ CONN_IN_PROGRESS
Threads	Provides summary information for the EDBC server. Max= Maximum number of gateway threads allowed. Curr= Number of active threads for this gateway. This includes both internal and gateway threads. User= Number of active gateway threads for this gateway. Inactv= Inactivity timeout value. If this parameter is set to a value > 0, then it indicates the length of time a gateway thread is idle before it is terminated by the gateway. If it is set to 0, no timeout occurs.	

Adding a New Gateway User

To enable a new user to access CA-IDMS data through the gateway, you must perform the following steps:

1. Define an OS/390 user ID with authorized access to TSO.
2. Define the user to CA-IDMS and grant the necessary permissions for the CA-IDMS Central Version.

   If the new user is a remote user, notify the EDBC system administrator on the client with the name and password for the authorization ID. These values must be entered in netu to give the user remote access to the gateway.
3. ADDIDMSU in SAMPLE.CNTL is a customized job to assist you in adding a new user definition to the IIUSER table. Member ADDIDSMU contains this job. You must edit the job to provide the appropriate column data for the new user definition.

See STEP050 of the S2 stage2 job for the column data for users that was defined through the stage1 macros. You may also want to refer to the stage1 macros IGWFDBA and IGWFUSER for column descriptions.

4. If the user will be accessing the gateway via TSO, modify the user's TSO logon procedure to include the libraries that are needed to access the gateway. (See the Customizing the TSO Logon Procedure section in the "Installing the Gateway" chapter.)

Presenting Additional Gateway Objects to a User

By default, the list of objects (tables, views, and so on) presented to EDBC users at connection time is limited to objects owned by:

- The EDBC user (user_name and default_schema in the IIUSER table)
- The user's DBA (dba_name in the IIUSER table)
- The system_owner specified at installation time (usually $EDBC)

Example1

IIUSER table entry for user EDBCUSR1:

user_name	**dba_name**	**default_schema**
EDBCUSR1	EDBCDBA1	EDBCUSR1

In this example, the list of objects presented to EDBCUSR1 at connection time consists of objects owned by EDBCUSR1, EDBCDBA1, and the system_owner.

The IIGWSCHEMA_SPACE table is used to expand the default list of objects. Each row in IIGWSCHEMA_SPACE consists of 2 columns, dba_name and schema_name. If dba_name is set to PUBLIC, every EDBC user is additionally presented with tables owned by schema_name. If dba_name is set to the name of an EDBC DBA, every user in this DBA's group is additionally presented with tables owned by schema_name.

Example 2

The IIUSER entry for user EDBCUSR1 is the same as in Example 1 but the following entries have been added to IIGWSCHEMA_SPACE:

dba_name	**schema_name**
PUBLIC	EDBCUSR2
PUBLIC	EDBCUSR3
EDBCDBA1	EDBCUSR4
EDBCDBA1	EDBCUSR5

In this example, EDBC presents EDBCUSR1 with additional objects owned by EDBCUSR2 and EDBCUSR3 (because objects owned by these schemas have been added to the PUBLIC group) and objects owned by EDBCUSR4 and EDBCUSR5 (because EDBCUSR1 belongs to the EDBCDBA1 group).

Note: Even though EDBC may present additional objects to a user at connection time, the user needs to have appropriate GRANT privileges to gain access to these objects.

Defining an Additional EDBC Server

There may be times when you need to create an additional EDBC server to CA-IDMS on the same OS/390 host for load balancing or resource control. You may also wish to create additional servers to access other CA-IDMS Central Versions. The following procedure describes how to use the IIVP to define the second EDBC server.

Using IIVP to Create a Second EDBC Server

Follow these steps to create a second EDBC server using the IIVP:

1. Define a second gateway subsystem name to OS/390.

   Each server must have a unique OS/390 subsystem defined.

2. Add this new subsystem name to the IEFSSNxx member in SYS1.PARMLIB.

   You must IPL OS/390 at the end of this procedure to activate this new subsystem.

3. Define the network changes needed to access this second server.

   - For SNA LU0 protocol server support:

     Define the VTAM APPL for this server. You can duplicate the existing VTAM APPL and change the name.

   - For TCP/IP protocol server support:

     Select the Port ID for this server. This Port ID must be different from the original Port ID assigned to the first server.

   - For CCI protocol server support:

     Select the CCI Product ID for this server. This value must be different from the original Product ID assigned to the first server.

4. Edit the stage1 input file IGWFSTGS in the dataset with the default name EDBC.V2R3.FILES.ASM.

Make the following changes to each SYSGEN statement:

- IGWFJOB: It is recommended that you change the JOBNAME= prefix so that the members that are placed into the STAGE2.CNTL dataset have unique names and do not replace previous members. Use the new installation ID as part of the prefix.
- IGWFIDMS: No changes required.
- IGWFINET: Change the following parameters as indicated:

  SUBSYS=
  Set this parameter to the value of the new gateway subsystem name.

  ID=
  Set this parameter to the value of the new II_INSTALLATION value.

  MAXUSER=
  Set this parameter to the maximum number of users to be allowed on the system.
- IGWFPSVR: Add a new IGWFPSVR statement corresponding to the network protocol parameters that will be needed to access the new server. Use the following guidelines for each parameter:

  TYPE=
  Set this to the type of protocol server that the new gateway will use.

  ACB=
  If the LU0 protocol server is being used, set this value to the new VTAM APPL defined for this server.

  PLU=
  If the KNET protocol server is being used, set this value to that of the KNET primary Logical Unit value.

  PORT=
  If the KNET protocol server is being used, set this value to that of the new port ID associated with this server.

  INSTALL=
  Set this value to that of the IGWFINET ID= value.

  APPLID=
  If the Interlink SNS/TCP protocol server is being used, set this value if access to the SNS/TCP subsystem is restricted by the application id.

PASSWORD=
If the Interlink SNS/TCP protocol server is being used, set this value if a password is required to interface to the SNS/TCP subsystem.

SYSID=
If the Interlink SNS/TCP protocol server is being used, set this value if the subsystem name of the SNS/TCP subsystem is different than the default of ACSS.

PRODID=
If the CCI protocol server is being used, set this value to that of the new CCI Product ID associated with this server.

- IGWFBLD: Make the following change to this SYSGEN statement:

  PRODUCTS=(EDBC)
  This specifies that the IIVP stage1 is to create a jobstream that defines a new server, but does not install any new gateway system catalogs in CA-IDMS.

- IGWFPIPE If the new server will interface to CA-IDMS using the LU62 driver, add a IGWFPIPE statement.

5. Save the IGWFSTGS member in the dataset with the default name EDBC.V2R3.FILES.ASM.

6. Edit the member IGWFSTGS in the dataset with the default name EDBC.V2R3.SAMPLE.CNTL and submit it for execution.

   This job creates the IIVP stage2 jobstream in a dataset with the default name EDBC.V2R3.STAGE2.CNTL.

7. Review the jobstream in this dataset and submit it for execution.

   The jobstream consists of the following jobs:

   – IGWFVPA0: Creates logical symbols to customize the gateway
   – IGWFVPI0: Customizes EDBC
   – IGWFVPN0: Creates name server files
   – IGWFVPP0: Customizes the CA-IDMS PIPE (LU62 interface only)
   – IGWFVPS4: Installs CA-IDMS cross memory programs (if IGWFPIPE was not specified)

8. Release each job sequentially.

   These jobs define a second EDBC server to CA-IDMS. See the Final Installation Procedures section in the "Installing the Gateway" chapter for the post-installation steps that you must complete.

Cloning an EDBC Server to a Different LPAR

The following procedure describes how to use the EDBC installation process to clone an existing EDBC server installation to a different LPAR(s):

1. Define the required subsystem(s) on the target LPAR.
2. APF authorize the EDBC server load libraries.
3. Make a copy of the customized stage1 for the existing installation.
4. Change the following stage1 macros:
   - IGWFJOB—Change the JOBNAME= parameter so that the members that are placed into the STAGE2.CNTL dataset have unique names and do not replace any existing members.
   - IGWFINET—Change the ID= parameter to a new II_INSTALLATION value.
   - IGWFPSVR—Change the USERID= parameter to the value for TC/PIP on the target LPAR.
5. Customize the appropriate DBMS macro(s) (that is, IGWFDB2, IGWFDCOM, IDMS, and so on).
6. Submit the stage1 job.
7. Submit the generated stage2 jobs.

Except for the IINAME files, EDBC data sets from the original install are shared across LPARs.

Connecting from One EDBC Server to Another

When you want to connect from one EDBC server to a second EDBC server, use the procedure described in this section.

Once you have created an additional EDBC server, the additional server (EDC2) must be defined to act as a client of the first server (EDBC). You must complete the following procedure in order to allow a remote client connected to the second EDBC server to connect to the first EDBC server.

1. Verify that both servers (EDBC and EDC2) have been started.
2. From ISPF option 6, issue the following command to set up the logical symbols to access the second server (EDC2).

   %iiparm isvr(*edc2*) prefix(edbc.v2r3)

   This TSO Clist allocates the IIPARM DD statement that contains the logical symbols that allow the TSO address space to connect to the EDC2 server.

3. Type the following to invoke the netu utility:

   netu

   The netu utility is used to define the information required by an *EDBC* client to connect to a remote *EDBC* node, in this instance, EDC2.

4. At the NETU prompt, type n to select Modify Node Entry and press Enter.

5. At the following prompt

   ```
   Enter operation (add, del, show):
   ```

   type **add** and press Enter.

6. Type the following responses to the following prompts that appear:

Netu Prompt	Response
`Enter Private or Global (P):`	p
`Enter the remote vnode name:`	EDBCUSER01
`Enter the network software type:`	TCP_IBM
`Enter the remote node address:`	*Internet Address*
`Enter the remote server listen address`	*port ID*

7. When the following message appears:

   ```
   Private: 1 row(s) added to the Server Registry
   ```

   Type the following to return to the netu menu:

   /*

8. At the NETU prompt, type a to select Modify Remote Authorization Entry.
9. Type the following responses to the following prompts that appear:

Netu Prompt	Response
`Enter Private or Global (P):`	p
`Enter the remote vnode name:`	EDBCUSER01
`Enter the remote User Name:`	edcusr1
`Enter the remote Password:`	xxxxxxx
`Repeat the remote Password:`	xxxxxxx

10. When the following message appears:

    ```
    Private: 1 row(s) added to the Server Registry
    ```

    Type the following to return to the netu menu:

    /*

11. Type e to exit from netu.

    When you successfully complete these steps, a vnode entry with a corresponding remote user ID entry is defined to access the EDBC server, EDBC.

    Verify that the remote user name is a valid CA-IDMS authorization ID and the associated password is valid.

    For more information about netu, see the *System Administrator's Guide*.

Verifying the Connection

To verify that you can connect to the first EDBC server (EDBC) from the second server (EDC2), complete the following steps.

1. Verify that both servers are started.
2. Access EDC2 from ISPF option 6 from a TSO terminal.

   %iiparm isvr(edc2) pre(edbc.v2r3)

3. Issue the following command to invoke the Terminal Monitor, assuming the CA-IDMS Central Version name is DSNT.

   sql edbcuser01::dsnt/idms

   The Terminal Monitor prompt should appear, indicating that you have successfully connected to the first gateway.

The gateway catalog, iidbconstants, can be used to verify that the correct user ID is being used to access the gateway. Issue the following command from the Terminal Monitor:

select * from iidbconstants\g

Using the sample values, the following response should appear.

USER_NAM	DBA_NAME	SYSTEM_O
EDCUSR1	EDCUSR1	$EDBC

(1 row)

The value in the USER_NAM column should match what was specified in the remote user name definition.

Chapter 8

Working with CA-IDMS Data

Your primary source of information about the EDBC user interfaces is EDBC documentation for the appropriate platform. For detailed information about CA-IDMS, see the CA-IDMS documentation.

This chapter does not summarize the information in these documentation sets. Instead, it addresses the aspects of working with EDBC and CA-IDMS that are particular to the gateway. This chapter:

- Explains how to connect to CA-IDMS from an EDBC user interface
- Describes how to create new CA-IDMS tables
- Summarizes how to access existing tables in CA-IDMS
- Shows the mapping between EDBC data types and CA-IDMS data types

The Database in EDBC and in CA-IDMS

An EDBC and a CA-IDMS relational database management system consist of one or more databases. EDBC system catalogs cover a single database, but CA-IDMS system catalogs can cover all databases in one Central Version (CV).

The system catalogs for the gateway can cover an entire CA-IDMS Central Version.

The interaction between the IDMS gateway and CA-IDMS differs depending on whether the gateway is running in the EDBC server address space or in a TSO or batch address space. When running in the EDBC server address space, the interaction is by means of a cross memory interface or an LU62 (APPC) API.

In the cross memory option, two programs, EDBCECOM and EDBCRSPD, are installed in the IDMS Central Version. EDBCECOM acts as a "driver," which enables the communication between the IDMS Central Version and the EDBC server.

The EDBCECOM program is initiated by the task code "EDBCssid," where "ssid" is the subsystem ID of the EDBC server associated with a particular instance of an EDBCECOM program. Thus, multiple EDBC servers can communicate with a single IDMS Central Version. The "EDBCssid" driver is defined as an IDMS startup Autotask which is managed by two additional programs, EDBCSTRT and EDBCSTOP. Both EDBCSTRT and EDBCSTOP accept the subsystem ID as an input parameter, that is, "EDBCSTRT ssid" and "EDBCSTOP ssid." These tasks alternatively start and stop EDBCECOM in the IDMS address space. The EDBCECOM program initiates an occurrence of the EDBCRSPD program for each active IDMS gateway thread. The EDBCRSPD program accepts and performs SQL queries and returns the results to the gateway thread, using buffers in ECSA and cross memory WAIT/POST.

With the LU62 (APPC) API option, a single program, RHDCRSPD, is installed on the IDMS Central Version. This program performs the same functions as EDBCRSPD, but the interaction with the IDMS gateway uses the IDMS LU62 Line Driver interface. In addition to the RHDCRSPD program, a LINE, PTERMs and LTERMs must be defined to the IDMS CV and ACBs and MODEENTs defined to VTAM.

Cross memory is the recommended option when the EDBC server and IDMS CV are operating in the same complex.

When the IDMS gateway runs in a TSO or batch address space, the Mini-CV interface is used to interface the gateway to IDMS. The //SYSCTL DD statement defines the IDMS CV the gateway will interact with.

Accessing a CA-IDMS Central Version

To access a CA-IDMS Central Version through the gateway you must:

The following section describes how to access a CA-IDMS Central Version from an EDBC client.

- Know the name of the CA-IDMS secondary catalog to which a connection is to be established.

  Note: A CA-IDMS secondary catalog is synonymous with database.

- Have been defined to that CA-IDMS Central Version (CV).
- Have access to a TSO account on the OS/390 system where the CA-IDMS Central Version resides.

Connecting to CA-IDMS

This section explains how to connect to a CA-IDMS Central Version from an EDBC client and locally from TSO.

Connecting to CA-IDMS Remotely from an EDBC Client

The syntax for accessing a remote CA-IDMS Central Version is:

command vnode::*dbname*/ **idms** [*with_clause*]

The following table describes these parameters:

Parameter	Syntax
command	EDBC command to connect to a server
vnode::	The virtual node name of the remote node on which the CA-IDMS Central Version is located (note the two colons). The *vnode* name is defined on the system where the EDBC interface resides, using the Netu utility.
dbname	The name of the CA-IDMS secondary catalog to use for the connection. To default to the secondary catalog that is set for the server, specify IDMS.
idms	The server class being accessed. The server class for the gateway is IDMS.
with_clause	The only supported with clause option is idms_ct_option.

Upper and lowercase entries are both acceptable.

For example, to access a CA-IDMS secondary dictionary named CA-IDMS on a remote node with the vnode name london, type:

```
sql london::idms/idms
```

The with idms_ct_option clause allows you to append CA-IDMS-specific parameters to all create table statements for a given connection. For example:

```
connect ... with idms_ct_option = 'in
     "default.areaname"'
```

The with idms_ct_option clause overrides the server logical that is set for idms_ct_option. You can override this logical on a create table statement. For more information, see the Creating Tables in a Non-Default Database Area section in this chapter.

Connecting to CA-IDMS Locally from TSO

To access a CA-IDMS Central Version locally from TSO, using the EDBC Terminal Monitor, type:

command dbname [*with_clause*]

where *command* is any system level command, and *dbname* is the name of the CA-IDMS secondary dictionary.

For example, to use the Terminal Monitor to access a CA-IDMS secondary dictionary named IDMS, on the same host computer (the IIPARM DD statement points to the system logical symbols that control the connection), type:

sql idms

Connecting to CA-IDMS from an EDBC Application

You can access a CA-IDMS Central Version using an application coded in a host language program that contains embedded SQL statements.

Every embedded OpenSQL application must include an explicit connect statement to connect the program to the database. The connect statement must be the first executable SQL statement in the program.

If the application program is accessing a remote CA-IDMS database, the connect statement has the following syntax:

connect *vnode::dbname* [*with_clause*]

For example:

```
exec sql connect 'techs::idms';
```

If the application program is accessing a local CA-IDMS database (the IIPARM DD statement points to the system logical symbols that control the connection), the connect statement has the following syntax:

connect *dbname* [*with_clause*]

For example:

```
exec sql connect 'idms';
```

Accessing Multiple CA-IDMS Central Versions

The gateway can connect to multiple CA-IDMS Central Versions. See the "Multiple Central Version Support" appendix for details on implementing multiple CV support.

Creating Tables in CA-IDMS

This section describes important characteristics of CA-IDMS tables and how to create tables in CA-IDMS.

Rules Governing CA-IDMS Tables

All tables in a CA-IDMS Central Version are governed by CA-IDMS rules and requirements. This is true whether the tables are created by an EDBC user through the gateway or by a CA-IDMS user with a standard CA-IDMS application.

In CA-IDMS, the schema owner is the owner of a table or view. Only the owner is allowed access to tables or views. The schema owner can grant other users permission to access owned table or views.

In CA-IDMS, whenever you create a table, it is automatically placed in the default area specified for the SCHEMA, unless you specify otherwise. You can override the default area using an OpenSQL with clause. For more information, see the Creating Tables in a Non-Default Database Area section in the "Working with CA-IDMS Data" chapter.

Case Sensitivity

Gateway users should be aware that database objects such as tables, views, and owner names are stored in the CA-IDMS catalogs in upper case. Therefore, these objects appear in upper case when they are viewed in the standard catalogs.

Creating Tables in CA-IDMS Through the Gateway

You can create a table in CA-IDMS through the gateway by:

- Entering the Terminal Monitor and invoking the OpenSQL create table or create table as select statement from the command line
- Incorporating the OpenSQL create table or create table as select statement into a custom application created with:
  - ODBC application
  - ADO or OLEDB application
- Using the OpenSQL direct execute immediate statement with the CA-IDMS create table statement, either from the Terminal Monitor or from an ESQL application program

Whenever you create a table through the gateway, you should issue a commit statement directly afterward. CA-IDMS places locks on resources like the system catalogs whenever you create or drop a table. When you issue a commit statement, CA-IDMS releases those locks, freeing system resources for others' use.

Creating Tables in a Non-Default Database Area

When you create a CA-IDMS table through the gateway, the table is automatically placed in a default database area. The default name for this database area is EDBCDBA.II-DDATA-AREA, but a different value may have been assigned during the installation process.

To improve the way your applications run against CA-IDMS, you may want to create tables in other database areas. There are three ways to designate the database area name in which a table will be created. You can:

- Define a server logical. This specifies the default database area name for every create table statement for every user connection with this EDBC server. For more information, see the Using Server Logicals section in the "Working with CA-IDMS Data" chapter.
- Specify a new default database area name in which you place all newly created tables. The new default database area name must already have been created. To substitute the new default for the original default, supply the following parameter on the with clause of the connect statement:

  connect ... **with idms_ct_option =** '**in** *databasesegment.areaname*'

  This overrides the installation-defined default database area name for the duration of the connection, and creates all tables in the newly specified database area name.
- Specify the database area name of your choice by adding an SQL with clause to the create table statement. For example,

  create table *tablename* ... **with idms_option = 'in**
  databasesegment.areaname'

  This overrides any default database area name that was defined either with the connect statement or at installation.

Using Server Logicals

The IDMS_CT_OPTION logical allows you to append IDMS-specific parameters to the create table statement. The supported forms of this logical are:

IDMS_CT_OPTION	= '*CA-IDMS* ***create table*** *parameter*'
IDMS_CT_OPTION1	= '*CA-IDMS* ***create table*** *parameter*'
IDMS_CT_OPTION2	= '*CA-IDMS* ***create table*** *parameter*'
IDMS_CT_OPTION3	= '*CA-IDMS* ***create table****parameter*'
IDMS_CT_OPTION4	= '*CA-IDMS* ***create table*** *parameter*'
IDMS_CT_OPTION5	= '*CA-IDMS* ***create table*** *parameter*'

IDMS_CI_OPTION	= '*CA-IDMS* ***create index*** *parameter*'
IDMS_CI_OPTION1	= '*CA-IDMS* ***create index*** *parameter*'
IDMS_CI_OPTION2	= '*CA-IDMS* ***create index*** *parameter*'
IDMS_CI_OPTION3	= '*CA-IDMS* ***create index*** *parameter*'
IDMS_CI_OPTION4	= '*CA-IDMS* ***create index*** *parameter*'
IDMS_CI_OPTION5	= '*CA-IDMS* ***create index*** *parameter*'

Table Names in CA-IDMS and EDBC

CA-IDMS stores database objects in upper case in its system catalogs. As a result, the names of CA-IDMS tables, views, and indexes are in upper case when accessed from an EDBC user interface.

CA-IDMS creates two-part table names. It automatically joins the name of the current schema to the name of the table to create a table name in the format *schema.tablename*.

If you do not qualify a table reference, CA-IDMS will qualify it using the default schema contained in the IIUSER table.

The default schema for a user is set by the IGWFUSER stage1 parameter.

Table Creation by the Gateway DBA

The gateway DBA can create new CA-IDMS tables for you that you can access through the gateway. These tables can be defined using a variety of CA-IDMS tools. Once the table is created, the gateway DBA must grant you the necessary access privileges for the table.

Accessing Existing Tables and Views in CA-IDMS

To access an existing table or view in CA-IDMS you must be authorized to connect to the appropriate dictionary. In addition,

- The table or view must be defined as public, or
- You must own the table or view yourself, or
- You must have been granted access privilege(s) to the table or view by the object owner

To use an EDBC application to access an existing table you do not own, you must specify the fully qualified table name, *Schema.tablename*.

Table Access Privileges

As described in the Accessing Existing Tables and Views in CA-IDMS section, any user who owns a table or view in CA-IDMS can grant other users permission to access those database objects. CA-IDMS recognizes the following access privileges:

- select: to read from a table or view
- insert: to add new rows to a table or view
- delete: to delete rows from a table or view
- update: to change existing data in a table or view

See the *CA-IDMS SQL Reference Guide* for information about definition privileges.

Access to CA-IDMS Network Definitions and Data

EDBC access to IDMS Network data is initiated by defining an SQL schema with the CREATE SCHEMA FOR NONSQL SCHEMA statement.

The records defined in the non-SQL schema can be accessed as tables in SQL DML and CREATE VIEW statements. Each record element is represented as a column except as noted in Record Structure Considerations section in the *IDMS SQL Reference Guide*.

The syntax for create schema is as follows:

create schema *schema_name*
        default area *segment_name.area_name*
        for nonsql schema *nonsql_schema_specification*

(Expansion of *nonsql_schema_specification*)

dictionary-name.nonsql-schema_name **version** *version number*
dbname database_name

The *dictionary-name* names the dictionary that contains the non-SQL-defined schema. If dictionary-name is not specified, the default is the dictionary to which the SQL session is connected.

The *database-name* identifies the database containing the data described by the non-SQL-defined schema. Database-name must be either a segment name or a database name that is defined in the database name table. If database-name is not specified, no database name is included in the definition of schema-name. At runtime, the database to which the SQL session is connected must include segments containing the areas defined by the non-SQL-defined schema.

There are two aspects to the EDBC server accessing IDMS Network data definitions and data:

1. To view the IDMS Network data definitions, the EDBC server must be connected to a database name that includes the SQL segments containing the SQL schema as well as the segments containing the dictionary that contains the non-SQL schema.
2. To access the IDMS Network data, the EDBC server must be connected to a database name that includes the segments containing the application data areas defined by the Network schema.

A typical IDMS Central Version will contain one or more SQL catalogs (DDLCAT, DDLCATX, and DDLCATLOD areas) and one or more Dictionaries (DDLDML, DDLDCLOD, etc. areas). By default, an IDMS Central Version installation will include a SYSTEM database name that includes the system Dictionary(SYSTEM) and a SYSDICT database name that includes the primary Catalog(SYSSQL).

By default, the EDBC server installation procedure creates a secondary Catalog (EDCSQL) and creates a database name that includes the EDCSQL catalog and the IDMS SYSTEM dictionary. The EDCSQL catalog contains all of the metadata necessary to support all client access through the EDBC server.

The default EDBC server installation therefore provides no access to user-defined SQL schemas or Network schemas defined in a dictionary other than the SYSTEM dictionary.

Most IDMS installations keep their Network schema definitions in a secondary dictionary (not SYSTEM). The EDBC server does not require the IDMS system dictionary, so the simplest way to view Network data through the EDBC server is to modify the default EDBC server database name to include the EDCSQL catalog and the appropriate secondary dictionary. SQL schemas can be created in the EDCSQL catalog that "map" to the Network schema names.

If the installation has an SQL catalog already containing SQL schemas that map the IDMS Network schemas, the existing SQL catalog can be used in place of the default EDCSQL catalog. Please note that installation job IGWFVPS2 contains the metadata required by the EDBC server. This job step must be executed against every secondary SQL catalog that is to be used by the EDBC server.

In summary, to successfully access both the data definitions and the data, the values specified for dictionary and dbname in the SQL schema created to access nonSQL IDMS data are critical. The nonSQL schema must be qualified by using the runtime database name as the dictionary in which this non-SQL schema is found.

At runtime, when data access or data definitions are desired, a session must explicitly connect to a database name in the CA-IDMS database name table. This database name must include both the dictionary segment containing the Network schema definition and the segments(s) where the data is actually stored. Lastly, the nonSQL schema must use this same database name as the 'dictionary' name qualifying the nonSQL schema name.

Dropping Tables Through the Gateway

You can use the SQL drop table statement to remove an individual table from a CA-IDMS Central Version. You can drop a CA-IDMS table only if you own the table.

Similar to creating a table, CA-IDMS places locks on resources such as the system catalogs whenever you drop a table. Issue a commit just after dropping the table to release the locks and free system resources for other users.

You must also include the cascade parameter in order to remove views of the table and any referential constraints. You can use either one of the following options with cascade:

- Use the SQL direct execute immediate statement with the CA-IDMS drop table statement as follows:

  ```
  direct execute immediate drop table abc cascade;
  ```

- Use the with option:

  ```
  drop table abc with idms_option = 'cascade'
  ```

SQL and the Gateway

SQL statements and commands are supported by the operating system on which the EDBC user is running. When you work with a foreign database through a gateway, you must code your applications using OpenSQL.

OpenSQL maximizes application portability. If you code an application to OpenSQL specifications, you can run it against:

- Ingres databases

- Gateways, such as the Oracle gateway

OpenSQL

For portability across database management systems, the gateway supports most OpenSQL statements. This section summarizes the OpenSQL statements that you can use in applications that are running against the gateway. For further information on the syntax and usage of the OpenSQL statements, see the *OpenSQLReference Guide* for your platform:

The following table describes the extensions to OpenSQL that are supported by the gateway:

OpenSQL Statement	**Restrictions**
close	None
commit	None
connect	The following option is not supported: identified by *username*
create index	None
create table	None
create view	None
delete	None
describe	None
disconnect	None
drop index	Must be used with the OpenSQL statement, direct execute immediate
drop table	None
drop view	None
execute	None
execute immediate	None
fetch	None
open	The gateway ignores the following option in the statement's syntax and sets a warning flag: open for read only

OpenSQL Statement	Restrictions
prepare	None
rollback	None
select	None
set	The gateway supports the autocommit option of this statement
update	None

Extensions to OpenSQL

The gateway supports two major extensions to OpenSQL: The with clause and the direct execute immediate statement. These extensions, which are described in the following sections, allow you to pass through parameters and statements that OpenSQL does not recognize.

Using these extensions to OpenSQL can increase the capabilities of your application or enhance its performance, as described in the following sections. However, it affects portability. Including CA-IDMS-specific syntax in your application limits its utility to CA-IDMS databases, because the application can no longer run against an EDBC database or a different gateway.

The With Clause

The with clause enables you to specify CA-IDMS-specific options in an OpenSQL statement. The gateway supports the with clause with the following OpenSQL statements:

- connect
- create index
- create table
- create view
- drop table
- drop view

Valid with clause options depend upon the particular OpenSQL statement, as described in the following table:

With Clause Option	OpenSQL Statement	Description
check option	create view	Places restrictions on updating or inserting into a view. See the *OpenSQLReference Guide* for more information.
idms_ct_option	connect	Allows you to specify one or more CA-IDMS-specific parameters to append to all subsequent create table statements in a given connection.
idms_option	create index create table create view drop table drop view	Allows you to specify CA-IDMS-specific parameters. You can issue this statement multiple times to specify multiple parameters.

For example, to connect to a CA-IDMS Central Version IDMS, setting the default area for table creation to EDBCDBA.II-DDATA-AREA, type:

```
connect 'mvs1::IDMS/IDMS' with idms_ct_option =
    'in EDBCDBA.II-DDATA-AREA'
```

For more information, see the Connecting to CA-IDMS Remotely from an EDBC User Interface and Creating Tables in a Non-Default Database Area sections of the “Working with CA-IDMS Data” chapter.

The Direct Execute Immediate Statement

The direct execute immediate statement allows you to append a CA-IDMS statement to the OpenSQL direct execute immediate statement. The gateway passes the statement on to CA-IDMS without processing it.

Note the following considerations:

- A direct execute immediate statement can return a status message or completion code, but cannot return data to the calling program.
- The gateway does no syntax checking or verification of direct execute immediate statements. The CA-IDMS statement is passed as a character string for processing by CA-IDMS.

For more information on the direct execute immediate statement, see the *OpenSQLReference Guide,* and the Gateway Query Handling and Syntax Errors sections of the “Optimizing and Troubleshooting” chapter.

Data Types and Utilities

The following topics are covered because the information differs from or extends the conventions outlined in the *OpenSQLReference Guide*:

- Date support
- CA-IDMS time and date values, and their mapping to OpenSQL data types
- Other data types and their mapping to CA-IDMS data types

Date Support

The gateway supports the OpenSQL date data type. This data type is mapped to the CA-IDMS timestamp data type and vice versa.

Date values are stored in Greenwich Mean Time (GMT) and are converted to local time using the II_TIMEZONE logical. For more information, see the Local Time Zone section of the "Configuring Net" chapter.

Date values can contain any valid date between January 1, 1582, and December 31, 2382.

CA-IDMS Date and Time Values

The gateway does not support the CA-IDMS data types of date and time. When you input date or time values from an EDBC user interface to CA-IDMS, the values are passed directly to CA-IDMS without translation. You must be certain, therefore, that the character string you enter for a CA-IDMS date or time value has the correct format for that CA-IDMS data type.

It is recommended that you restrict your use of CA-IDMS date and time data types to display only. The way the gateway handles this data may change, and the usage may not be strictly upwardly compatible. If you confine this data to display only, you minimize the effect potential changes may have on your applications.

Note the following considerations:

- Date and time data from CA-IDMS tables are returned as character data and are translated into strings of 10 and 8 characters, respectively.
- You can supply new values or update existing values if the character string you enter for the date or time value has the correct format for that CA-IDMS data type.
- Updates are not successful unless the new string value represents a valid date or time value.

- In the following respects, this CA-IDMS data does not behave like standard character data:
  - If the column is used in a where clause, the comparison is done on date values, not on character string values.
  - If the column is used in an order by clause, the rows are sorted in time order rather than with the normal character string collating sequence.

Data Types

For portability across database management systems, the gateway supports the following OpenSQL data types:

- Character data: char and varchar
- Numeric data: smallint, integer, and float
- Date data: date

The gateway does not support data types such as integer1 and money.

Data Type Conversion

The gateway is responsible for all data type conversions necessary for access to CA-IDMS data and performs the following types of conversions:

- When you retrieve data from a CA-IDMS Central Version, the gateway converts it to an OpenSQL data type. This allows the EDBC user interface to interpret the data properly.
- When you create new data or update existing data, the gateway converts the input from the EDBC user interface into the corresponding CA-IDMS data type.

If there is no compatible data type, the query is rejected. If the new data type is not completely equivalent to the original, data type conversion may result in loss of precision or range.

CA-IDMS to OpenSQL Data Type Mapping

The following table lists the CA-IDMS data types that the gateway supports, and the OpenSQL data types to which they are converted. The table also indicates any restrictions on range or precision that the conversion imposes:

CA-IDMS Data Type	Open SQL Data Type	Restrictions
binary	Char	
char	Char	Maximum length in OpenSQL 2000
varchar	Varchar	Maximum length in OpenSQL 2000

CA-IDMS Data Type	Open SQL Data Type	Restrictions
date	char	Length 10
Decimal*	float(8)	
Double precision	float(8)	
Float (single precision)	float(4)	
Float (double precision)	float(8)	
Graphic		Not supported
Vargraphic		Not supported
integer	integer	
long integer	float(8)	
numeric	float(8)	
smallint	smallint	
real	float(4)	
time	char	Length 8
Timestamp	date	Length 26
Unsigned numeric		Supported
Unsigned decimal		Supported

* **Note:** CA-IDMS permits you to define decimal data in CA-IDMS tables. When a query to CA-IDMS returns decimal data, the CA-IDMS gateway maps this value into float or float8 columns. Thus, when viewed from an EDBC user interface, columns defined as decimal values in CA-IDMS appear as float columns instead.Decimal data can be returned as char if desired. See the IDMS_DECIMAL_AS_CHAR system logical in the "Gateway Logical Symbols and the IIPARM Clist" appendix.

OpenSQL to CA-IDMS Data Type Mapping

The following table lists the OpenSQL data types and the CA-IDMS data types to which they are converted, with relevant restrictions on range or precision:

OpenSQL Data Type	CA-IDMS Data Type	Restrictions
char	Char	Maximum length in OpenSQL of 2000
varchar	Varchar	Maximum length in OpenSQL of 2000
smallint	Smallint	-32,768 to +32,767
integer	integer	-2,147,483,648 to +2,147,483,647
float	float (double precision)	None
date	timestamp	None
time		Not supported

Using the EDBC Terminal Monitor

When you run the EDBC Terminal Monitor against the gateway, it is recommended that you do so in autocommit on mode. This causes the gateway to treat each statement as a transaction, and to issue an implicit commit after each statement.

This is desirable because when a statement such as create or drop is issued to the gateway, locks are granted in CA-IDMS. These locks are held until you issue a commit or a rollback. The set autocommit on statement causes an implicit commit to occur after each query, thus releasing any locked resources. So, running the Terminal Monitor in autocommit on mode improves concurrency and prevents timeouts from CA-IDMS.

The default setting for the Terminal Monitor is autocommit off. You can change this setting in either of the following ways:

- Interactively, each time you enter the EDBC Terminal Monitor (SQL).
- By setting the gateway logical ING_SET to set autocommit on.

  This changes the default setting. You set the logical ING_SET where you invoke the Terminal Monitor, not on the system where the gateway is running.

Chapter 9

Using Database Procedures

The CA-IDMS gateway supports database procedures that allow remote users to invoke user-written procedures on the host operating system.

This chapter addresses the aspects of working with database procedures that are particular to the CA-IDMS gateway. This chapter:

- Summarizes the input parameters provided to database procedures
- Outlines the coding conventions for database procedures
- Describes input parameters
- Describes non-function call macros
- Describes the database procedure function call macros
- Outlines data handling
- Explains how to register, execute, and remove database procedures
- Discusses the verification process for database procedures
- Discusses how to use EXCI

Database Procedures Defined

Database procedures are application programs that can be invoked through the CA-IDMS gateway. These programs must be written in Assembler.

Database procedures allow access to CICS transactions using the EXCI interface.

Select Procedures and Message Procedures

Database procedures operate in one of the two following modes:

- *Select mode* which allows a client to receive formatted data (rows) as output from the procedure.
- *Message mode* which only allows unformatted data (messages) to be sent to the client.

Select mode operation requires that you:

- Register the procedure as a SELECT_PROCEDURE or FUNCTION.
- Define output fields using SELECT_OUT or OUTPUT to get an output SQLDA built automatically, or the procedure can set up its own output SQLDA.
- Execute the procedure to get the input and output SQLDAs set up with client supplied input data or defaults.
- Select the procedure to actually invoke the user procedure code.

Message mode operation requires that you:

- Register the procedure.
- Execute the procedure.

Coding Database Procedures

The following sections describe coding database procedures. CA-IDMS database procedures must be written in IBM assembler language. The programs are required to be re-entrant and to run in address mode 31.

Coding Conventions

The interface program (SASMINT) is coded using IBM Standard Assembler conventions.

The SASMxxxx members in the &prefix.SAMPLE.ASM library should be used as a starting point for new routines.

The general order of using the supplied DBPxxxxx macros should be:

- DBPPROLG should be the first statement in order to copy in and have the function call macros defined inline to the program.
- Save area handling and input parameter USINGs would be next.
- A mainline routine that invokes processing subroutines is next.
- On return, the RETVALUE field being non-zero will cause a message to be sent back to the user. A 1 or 2 will cause any message put into the TEXT field to be sent out to the user. Any other non-zero value will cause a message of 'RAISE ERROR 999' to be sent out.
- Any constants could be placed here.

- Processing subroutines could be placed here. These would use whatever function call macros that are needed, for example, ALLOC_MEMORY, FREE_MEMORY, GET/PUT_VARIABLE, CALL_EXCI, SEND_DESCRIPTOR, SEND_DATA for rows and SEND_DATA for 'all rows sent'. Each subroutine could be covered by a secondary base register if needed.
- Then DBPWORKI would be used to define the dynamic area required for a save area, parm pointer and call area. Any user required items (up to a 4096 byte overall limit) could be added also.
- The DBREGS macro could come here or after the DBPPROLG macro.
- The DBPEPILI macro would define the required DSECTs for the input parameters, the function list and the SQLDA structures.
- END of program.

Library Considerations

The database procedure must be re-entrant, address mode 31 and linked into the //IIUSRLIB DD file. The supplied data set is &prefix.USER.LOAD. This library is not required to be APF authorized.

Procedure Completion

You can set the return_value and return_text parameters to return to the client if an error occurs. Return_text must be followed by a null byte.

When your procedure completes, it must return one of the following codes to the CA-IDMS gateway:

- 0 if the procedure completed successfully
- 1 is undefined for CA-IDMS
- 2 if a procedure error occurred

Note: Return_text will be set in the alloc_memory, get_variable, put_variable, and the send_data routines if there is an error.

The return_text is sent back to the client if the return code is 1 or 2. A return code that is not 0, 1, or 2 produces a RAISE ERROR 999 message.

Input Parameters

The DBPEPILI macro defines DSECTs that are used to map out all of the parameters that are passed to the Database Procedure.

PARMLIST is the base DSECT. It consists of pointers to:

ISQLDA	DS AL4	The Input SQLDA
OSQLDA	DS AL4	The Output SQLDA
OSQLCA	DS AL4	The SQLCA work area
RETVALUE	DS AL4	The return value (completion code) area
TEXT	DS AL4	The completion message area
DBFUNC	DS AL4	The function list
TOKEN	DS AL4	A token to passed to the interface routine when called by one of the support functions.
SAVEWRK	DS AL4	A 4096 byte save/work area for the procedure.

DBFUNC points to a list of callable support routine addresses and is covered by the FUNCLIST DSECT.

ENDTX	Use the END_TRANSACTION macro to invoke this routine.
ALLOCMEM	Use the ALLOC_MEMORY macro to invoke this routine.
FREEMEM	Use the FREE_MEMORY macro to invoke this routine.
SENDMSG	Use the SEND_MESSAGE macro to invoke this routine.
CALLEXCI	Use the CALL_EXCI macro to invoke this routine.
SENDDESC	Use the SEND_DESCRIPTOR macro to invoke this routine.
SENDDATA	Use the SEND_DATA macro to invoke this routine.
PRCTRACE	Use the PROC_TRACE macro to invoke this routine.
GETVAR	Use the GET_VARIABLE macro to invoke this routine.
PUTVAR	Use the PUT_VARIABLE macro to invoke this routine.

ISQLDA points to the Input SQLDA and is covered by the INSQLDA DSECT.

This SQLDA structure contains an entry for each registered variable in the order they were registered. Another DSECT (INSQLVARN), can be used to cover the metadata structure for each variable. INSQLDATAV can be used to cover the input DATAVARN structure.

OSQLDA points to the Output SQLDA and is covered by the OUTSQLDA DSECT.

This SQLDA structure contains an entry for each variable that was registered as an OUTPUT variable. Another DSECT (OUTSQLVARN), can be used to cover the metadata structure for each variable. OUTSQLDATAV can be used to cover the output DATAVARN structure.

This SQLDA is the one that is used to send rows (tuples) back to the client. The first thing sent to the client must be a tuple descriptor. This is done via the SEND_DESCRIPTOR macro. Then as many SEND_DATAs can be done as needed to send data rows to the client. This must be followed by a special version of the SEND_DATA that indicates 'all rows sent'.

Assembler Macros

There are a number of members that are supplied in the FILES.MACLIB library to support writing Database Procedures. The following list introduces the members. An expanded explanation follows where needed:

DBPALLOC	Contains the ALLOC_MEMORY macro.
DBPCALL	Internal program call macro.
DBPCEXCI	Contains the CALL_EXCI macro.
DBPC2HEX	Contains the CONVERT_TO_HEX macro.
DBPDFMSG	Contains the DEFINE_MESSAGE macro.
DBPENDTX	Contains the END_TRANSACTION macro.
DBPEPILI	Defines PARMs, FUNCTIONs and SQLDAs
DBPEXCI	Defines the EXCI control structure.
DBPFREEM	Contains the FREE_MEMORY macro.
DBPGTVAR	Contains the various get data macros.
DBPPROLG	Macro to copy in others and set the CSECT name.
DBPPRTRC	Contains the PROC_TRACE macro.
DBPREGS	Defines register equates.
DBPSNDAT	Contains the SEND_DATA macro.
DBPSNDES	Contains the SEND_DESCRIPTOR macro.
DBPSNDMS	Contains the SEND_MESSAGE macro.
DBPSQLDI	Defines SQLDA structures
DBPUTVAR	Contains the various put data macros.
DBPWORKI	Defines the save area, parm pointer and call areas.

Generated Error Messages

If there is an error during the execution of ALLOC_MEMORY, GET_VARIABLE, PUT_VARIABLE or SEND_DATA, a message is put into the TEXT parameter area that explains what happened. A return code of 1 or 2 will cause this message to be sent back to the client. A PROC_TRACE macro could be used to send it to the IIGWTR trace file.

Other Notes

When invoking the interface programs, each of the macros supplies a TOKEN as the last parameter. This is done automatically. The TOKEN is an input parameter.

No data conversions are done during any of the GET/PUT variable processes. The data must be in the appropriate format.

Non-Function Call Macros

The following macros are required but do not perform any function calls:

DBPPROLG

This macro is used to start off a Database procedure. It will copy in other library members to define the function macros inline with an optional listing. The label on this macro will become the CSECT name for the module. A TITLE will be generated.

name	name: symbol. The name must begin in column 1.
b	One or more blanks.
DBPPROLG	
b	One or more blanks.
—	
title	title: literal. The TITLE for the program.
, LIST=list	list: YES or NO (default)

name
: This is a required label to give the programs CSECT name.

title
: A literal in single quotes that will be the TITLE for the program.

,LIST=
Specifies if the copied macros should be produced in the program listing.
NO is the default.

Example:

DBPROC DBPPROLG 'Title for the program',LIST=YES

DPBREGS

This macro is used to define equates for the registers.

b	One or more blanks
DBPREGS	
b	One or more blanks.

Example:

```
DBPREGS
```

DBPWORKI

This macro defines the dynamic work area for the program. A 4096 byte area is passed to the program that can be used as the dynamic area.

The items defined in this area are an 18-word save area, a word to save the input parms pointer, and an 8-word function call area. User items may be placed immediately following the macro to have them included in the dynamic area. A DSECT with a label of WORK is generated.

b	One or more blanks
DBPWORKI	
b	One or more blanks.

Example:

```
DBPWORKI
```

DBPSQLDI

This macro generates the DSECTs for the SQLDA structure.

SQLDA covers the base section.

SQLVARN covers the SQL variable metadata information.

DATAVARN covers the data/null indicator pointer structure.

b	One or more blanks.
DBPSQLDA	
b	One or more blanks.

Example:

```
DBPSQLDA
```

DBPEXCI

This macro defines the structures needed to access CICS through the EXCI interface.

b	One or more blanks.
DBPEXCI	
b	One or more blanks.

Example:

```
DBPEXCI
```

DBPEPILI

This macro is used to define the DSECTs that cover the input structures. The function macros will use these DSECTs in their generated statements. The following DSECTs are generated:

DSECT Name	Description
PARMLIST	Covers the input parameter list
FUNCLIST	Covers the function list pointed to by the input parm DBFUNC.
INSQLDA	Covers the Input SQLDA pointed to by the input parm ISQLDA.
INSQLDATAV	Covers the DATAVARN section of the input SQLDA.
INSQLVARN	Covers the SQLVARN section of the Input SQLDA
OUTSQLDA	Covers the output SQLDA pointed to by the input parm OSQLDA.
OUTSQLDATAV	Covers the DATAVARN section of the output SQLDA.
OUTSQLVARN	Covers the SQLVARN section of the output SQLDA.

b	One or more blanks.
DBPEPILI	
b	One or more blanks.

Example:

```
        DBPEPILI
```

Function Call Macros

The following macros are used to invoke the supplied functions:

ALLOC_MEMORY

Used to get dynamic storage for use during the procedure execution. Check for a non-zero return code. An error message will have been put into TEXT if an error occurred.

—

name	name: symbol. Begin name in column 1
b	One or more blanks
ALLOC_MEMORY	
b	One or more blanks.
memsize	memsize: decimal, register (2)-(12), or RS-type address
,memptr	memptr: A-type address or register (2)-(12)

—

MEMSIZE
: This specifies the amount of memory to be allocated. The total amount of memory that can be allocated is controlled by a System Logical in the IIPARM file. (The logical is IDMS_DBPROCS_MEMORY.)

,MEMPTR
: This is the storage location to receive the address of the allocated memory.

Example to allocate 512 bytes of memory:

```
         ALLOC_MEMORY   MEMSIZE,MEMPTR
         LTR      R15,R15
         BNZ      ALLOCERR

MEMSIZE  DC   F'512'
MEMPTR   DS   A
```

FREE_MEMORY

Used to free memory obtained via the ALLOC_MEMORY macro. Check for a non-zero return code. **Do Not Do Partial Frees**. Always free the entire amount. The interface program will free any areas that are not explicitly freed and send a message to the IIERLOG file.

----

name	name: symbol. Begin name in column 1
b	One or more blanks

FREE_MEMORY	
b	One or more blanks.
memsize	memsize: decimal, register (2)-(12), or RS-type address
,memptr	memptr: A-type address or register (2)-(12)

—

MEMSIZE

This specifies the amount of memory to be freed.

,MEMPTR

This is the storage location that points to the address of the allocated memory.

Example to free 512 bytes pointed to by register 2:

```
FREE_MEMORY   512,(R2)
LTR    R15,R15
BNZ      FREEERR
```

END_TRANSACTION

Used to indicate that a DBMS commit or rollback has been done. This macro should not be used if no DBMS calls are performed.

name	name: symbol. Begin name in column 1
b	One or more blanks
END_TRANSACTION	
b	One or more blanks.

Example:

```
END_TRANSACTION
```

DEFINE_MESSAGE

Used to set up a null terminated message that can be sent to the client via a SEND_MESSAGE or to the trace file via a PROC_TRACE.

name	name: symbol. Begin name in column 1.
b	One or more blanks.
DEFINE_MESSAGE	
b	One or more blanks
MSG='msg'	msg: The message to be defined.
,LEN=len	len: decimal digit. Optional len of message.

—

MSG=

The string of characters that make up the message within single quotes.

,LEN=

An optional length for the message. Could be used to set a standard length.

If a 'name' is specified, a symbol for the overall length (including null) is generated in the form of 'NAME#L'.

Example:

```
DEFMSG          DEFINE_MESSAGE LEN=30,MSG='A defined_message'
```

SEND_MESSAGE

Used to send a message back to the client.

Note: Messages are sent back as Error transactions. ODBC clients can not handle data sent back to the client in this manner.

name	name: symbol. Begin the name in column 1.
b	One or more blanks

SEND_MESSAGE

b	One or more blanks.
code	code: decimal digit
,msg	msg: literal, RX-type address or register (2)-(12)

—

code

An optional code that could be used as an identifier. Defaults to 0.

,msg

The message to be sent out to the client.

Examples:

```
SEND_MESSAGE   0,'Message from a literal'
SEND_MESSAGE   0,TMMSG
SEND_MESSAGE   0,TMMSG1
LTR  R15,R15
BNZ MSGERR
```

```
TMMSG   DC  C'Message from a DC',X'00'     Null terminated
TMMSG1 DEFINE_MESSAGE  LEN=30,MSG='A define_message message'
```

PROC_TRACE

Used to send a trace message to the gateways IIGWTR trace file.

name	name: symbol. Begin name in column 1.
b	One or more blanks.
PROC_TRACE	
b	One or more blanks.
msg	msg: RX-type of address

—

msg
: Address of the null terminated message that is to be sent to the IIGWTR trace file.

Example:

```
        PROC_TRACE   TRACE_MSG

TRACE_MSG   DC   C'This is a trace file message',X'00'    Null terminated
```

CALL_EXCI

Used to communicate with a CICS system with EXCI. The DBPEXCI control block must be set up prior to this. Refer to the sample EXCI program (SASMECIT) to see how this is done.

name	name: symbol. Begin name in column 1.
b	One or more blanks.
CALL_EXCI	
b	One or more blanks.

There are no direct parameters for this macro.

It assumes that the DBPEXCI control structure is addressable and has been filled in properly.

Example:

```
        CALL_EXCI
        LTR   R15,R15
        BNZ  EXCIERR
```

SEND_DESCRIPTOR

Used to send out a tuple descriptor to the client. This defines the rows (tuples) that will be sent to the client via the SEND_DATA macros that send out the data. Do not continue with SEND_DATAs unless you receive a zero return code from this process.

name	name: symbol. Begin name in column 1

b	One or more blanks
SEND_DESCRIPTOR	
b	One or more blanks.
SQLDA=addr	addr: A-type address Default: OSQLDA

—

SQLDA=
: Gives the address of the pointer to the SQLDA to use for sending out data rows to the client. This defaults to the Output SQLDA (OSQLDA).

Example of sending the descriptor:

```
SEND_DESCRIPTOR   SQLDA=OSQLDA
LTR  R15,R15
BNZ  DESCERR
```

OSQLDA is the Output SQLDA that is an input to the program.

SEND_DATA

Used to send an individual row back to the client or to indicate that all rows have been sent. Always issue an 'all rows sent' call, even after an error on a row send. An error message will have been put into TEXT if an error occurred.

name	name: symbol. Begin name in column 1
b	One or more blanks
SEND_DATA	
b	One or more blanks
0 1	dataend: 0 = no, send out another row 1 = yes, 'all rows sent'
,SQLDA=addr	addr: A-type address Default: OSQLDA

dataend

Indicates whether this call is sending a row or indicating that all rows have been sent.

,SQLDA
: Gives the address of the pointer to the SQLDA to use for sending out data rows to the client. This defaults to the Output SQLDA (OSQLDA) pointer.

Example of sending one row:

```
         SEND_DATA   0,SQLDA=OSQLDA    send a row
*  No test for error because program falls through to 'all rows sent'
         SEND_DATA   1,SQLDA=OSQLDA    indicate all rows sent
         LTR R15,R15
         BNZ DATAERR
```

OSQLDA is the Output SQLDA that is an input to the program.

GET_VARIABLE

Used to retrieve a field from an SQLDA structure. The data will be returned to the program. The LEN limits the amount to be returned and must be large enough to accommodate the field. Any errors (indicated by a non-zero return code) will have a message put into the TEXT (a program input parameter) message area. The variable can be retrieved either by name or index (relative to 0) position.

name	name: symbol. Begin name in column 1.
b	One or more blanks
GET_VARIABLE	
b	One or more blanks.
LEN=len	len: symbol, decimal or register (2)-(12)
,ADDR=addr	addr: RX-type address or register (2)-(12)
,NAME=name	name: RX-type address or register (2)-(12)
,INDEX=index	index: symbol, decimal digit, or register (2)-(12)
,SQLDA=sqlda	sqlda: A-type address or register (2)-(12) Default: ISQLDA

LEN=

Specifies the length of the receiving field. Use of register notation means that the length is contained in the specified register. An error will be generated if this is not large enough to hold the field. An error message will be placed into the TEXT area. The actual length of the data will be placed into the call area parameter GETVLEN. It will be zero for a null field.

,ADDR=

Specifies the address of the receiving field.

,NAME=

Name is the pointer to a location or a register notation for a register that contains the address of the location. The format of the location is a half-word length followed by the value of the name. The maximum length of a name for CA-IDMS is 32.

```
NAME   DC   H’16’,CL16’VAR_SIXTEEN’
```

,INDEX=

Index is the value (relative to 0) or a register notation for a register that contains the value for the variable to be retrieved.

,SQLDA=

The pointer to the SQLDA structure that the data is to be retrieved from. The default is the Input SQLDA (ISQLDA).

In the call area, a field (DBPGVVAR) will contain the pointer to the SQLVARN for the variable upon successful completion. DBPGVDAT will also be set for the pointer of the DATAVARN for this variable. These fields could be copied into others for use later on to directly access the metadata, data, or null indicators.

Example to get the second variable (the two macros are equivalent since INAME1 is the name of the second variable):

```
        GET_VARIABLE   LEN=64,ADDR=GVDATA,NAME=INAME1
        GET_VARIABLE   LEN=64,ADDR=GVDATA,INDEX=1
        LTR  R15,R15
        BNZ  GETERR

INAME1   DC   H’12’,C’VAR_NUMBER_2’
```

PUT_VARIABLE

Used to place data into an SQLDA (normally the output one) to have it sent back to the client. Check the return code after execution. If it is non-zero, an error message will have been placed into the TEXT message area.

name	name: symbol. Begin name in column 1.
b	One or more blanks
PUT_VARIABLE	
b	One or more blanks.
LEN=len	len: symbol, decimal or register (2)-(12)
,ADDR=addr	addr: RX-type address or register (2)-(12)
,NAME=name	name: RX-type address or register (2)-(12)
,INDEX=index	index: symbol, decimal digit, or register (2)-(12)
,SQLDA=sqlda	sqlda: A-type address or register (2)-(12) Default: OSQLDA

—

LEN=
Specifies the length of the variable field. Use of register notation means that the length is contained in the specified register. An error will be generated if this is not large enough to hold the field. An error message will be placed into the TEXT area. It will be zero for a null field.

,ADDR=
Specifies the address of the variable field.

,NAME=
Name is the pointer to a location or a register notation for a register that contains the address of the location. The format of the location is a half-word length followed by the value of the name. The maximum length of a name for CA-IDMS is 32.

```
NAME   DC   H'16',CL16'VAR_SIXTEEN'
```

,INDEX=
Index is the value (relative to 0) or a register notation for a register that contains the value for the variable.

,SQLDA=
Specifies the pointer to what SQLDA structure the data is to be put into. The Output SQLDA (OSQLDA) is the default.

In the call area, a field (DBPPVVAR) will contain the pointer to the SQLVARN for the variable upon successful completion. DBPPVDAT will also be set for the pointer of the DATAVARN for this variable. These fields could be copied into others for use later on to directly access the metadata, data, or null indicators.

Example:

```
        PUT_VARIABLE  LEN=4,ADDR=INTEGER,NAME=COST
        LA   R2,4
        PUT_VARIABLE  LEN=(R2),ADDR=INTEGER,INDEX=2
        LTR  R15,R15
        BNZ  PUTERR

COST    DC  H'4',C'COST'
```

GET_IDMS_VARIABLES

Used to get a set of non-null variables from an SQLDA starting with the first variable. There is no call to the interface program. This macro expands inline for each variable. This is handy to take a tuple data area and to place it into a non-tuple data area. This goes directly into the Input SQLDA. This macro uses R0, R1, R14 and R15.

```
GET_IDMS_VARIABLES VAR0,VAR1,VAR2,VAR3,VAR4,VAR5,VAR
```

Each VARx is the address of variable to receive data from the Input SQLDA.

Example—where R3 is pointing to a contiguous area to retrieve a 10 byte field, a 20 byte field, a 6 byte field, a 4 byte field, and a 40 byte field:

```
        LA    R3,RECORD
        GET_OUTPUT_VARIABLES  0(R3),10(R3),30(R3),36(R3),40(R3)
```

PUT_IDMS_VARIABLES

Used to put a set of non-null variables into an SQLDA starting with the first variable. There is no call to the interface program. This macro expands inline for each variable.

This is handy to take a non-tuple data area and to place it into a tuple data area. This goes directly into the Output SQLDA. This macro uses R0, R1, R14, and R15. R2 is also used for CA-Datacom/DB.

```
    PUT_IDMS_VARIABLES VAR0,VAR1,VAR2,VAR3,VAR4,VAR5,VAR6
```

Each VARx is the address of variable to be put into the Output SQLDA.

Example—to put the same fields retrieved with the above GET_OUTPUT_VARIABLES:

```
LA   R3,RECORD
PUT_OUTPUT_VARIABLES  0(R3),10(R3),30(R3),36(R3),40(R3)
```

Data Handling in Assembler

The input and output SQLDAs contain metadata about the data along with pointers to the data and to null indicators.

An important point to remember, is that the data for a tuple is not just a contiguous set of bytes necessarily. Access to the data portion should be done through supplied macros or by using the data pointers from the SQLDA and associated metadata.

There are three parts to the SQLDA (macro DBPSQLDI).

1. The SQLDA base section. This contains allocated and used counts for the number of elements contained in the SQLVARN. It also contains a pointer to the DATAVARN section.
2. The SQLVARN section (immediately follows the base section) contains the metadata for each used data item (field, column).
3. The DATAVARN section contains a pointer to the data and a pointer to the null indicator for each used data item.

If you use the GET_VARIABLE macro, the DBPGVVAR field defined using the DBPWORKI macro will contain a pointer to the SQLVARN entry for the associated variable on completion. Also, the DBGVDAT field will point to the DATAVARN entry. This macro uses the input SQLDA by default. SECHOPRC in the SAMPLE.ASM library uses the GET_VARIABLE macro.

If you use the PUT_VARIABLE macro, the DBPPVVAR field defined using the DBPWORKI macro will contain a pointer to the SQLVARN entry for the associated variable on completion. Also, the DBPPVDAT field will point to the DATAVARN entry. This macro uses the output SQLDA by default.

The GET_OUTPUT_VARIABLES and the PUT_OUTPUT_VARIABLES macros can be used to get or put more than one non-null field at a time starting with the first field.

SASMECIT in the SAMPLE.ASM library uses the PUT_OUTPUT_VARIABLES macro.

Implementing Database Procedures

To implement a database procedure, you need to perform the following steps:

In this procedure, the output of one step becomes the input of the next step.

1. Code the database procedure using one of the appropriate sample procedures supplied in the following sections.
2. Compile (assemble) the database procedure to produce an object deck that will be input into the next step.
3. Link edit the object deck and pull in any referenced language specific routines along with the required language specific interface module.

   The external reference in the interface module must be changed (by means of a linkedit statement) to reference the new database procedure code.
4. Register the database procedure with the gateway.

   See the Register Procedure Statement section for more information.

Steps 2 and 3 are performed by a single job using a customized job that is placed into USER.CNTL during stage1/stage2 processing. See the Assembler Database Procedures section.

Step 4 is run through the gateway. The sample procedures use SQL for the Terminal Monitor program to perform the register and test the procedure. The Terminal Monitor SQL is located in the FILES.SQL library. SAMPLE.CNTL members contain the JCL necessary to run the Terminal Monitor program.

Assembler Database Procedures

Database procedures written in Assembler can follow the standard IBM 370 coding conventions. An interface program (SASMINT) and a set of macros (DBPxxxxx) are provided to help you develop these procedures. The "Database Procedure Examples" appendix contains an annotated sample Assembler database procedure and describes the available macros.

Code the database procedure using one of the following supplied sample procedures as a starting point. The supplied samples are:

SASMECIT	A sample that returns rows from a CICS transaction invoked by means of the CICS EXCI interface.
SASMTST	A sample that shows the use of the DBPxxxxx macros.

SECHOPRC	A sample that simply echoes the values that come into the database procedure. The input SQLDA is used for this as opposed to the output SQLDA which is normally sent out by means of the SEND_DESC and SEND_DATAs. The input SQLDA contains both the registered input and output fields. This sample allows the user to test the way fields are registered to assure that the defaults desired are set up properly.

Implementing an Assembler database procedure involves using USER.CNTL(SASMIDMN), the customized job that performs the compile and link edit. This job accomplishes Steps 2-3 of the implementation as described in the Implementing Database Procedures section. (Make the noted modifications before submitting the JCL.)

You must register the database procedure with the gateway. (Registrations for the supplied samples are contained in the FILES.SQL library with a member name equal to the above listed samples.)

Coding Database Procedures

The following sections describe coding database procedures.

Coding Conventions

The IBM Standard Assembler SASMINT interface program is supplied with the CA-IDMS gateway.

The interface program allows the database procedure to be coded in the normal language conventions. This avoids the problem of using an unfamiliar convention as was required for Assembler in previous releases. Also, any earlier Whitesmith C procedures can cause problems if not handled correctly (no globals and no main). It is recommended that any new database procedures use the provided interface programs and follow the normal coding conventions for the chosen language. The examples provided in SASMxxx in the SAMPLE.XXX libraries should be used as starting points.

APF Authorization

In order to execute database procedures, the procedures must reside in an APF-authorized load library. The library must be in either the system linklist or the gateway server procedure STEPLIB or JOBLIB.

Optionally, if a library is provided by means of the IIUSRLIB DD statement, this library can be unauthorized and any database procedures in the library can be executed by the gateway.

Registering Database Procedures

This section describes the register procedure statement and the permissions required to issue it.

Register Procedure Statement

To register a procedure with a server so that it can be invoked by the execute procedure statement, use the following syntax:

register procedure *proc_name*
  (*param_nameformat qualifiers{,param_name format qualifiers***)**
  as [**import**|**select_procedure**|**function**] [**from**'source']

Parameter	Description
proc_name	Specifies the procedure name.
param_name	Specifies the input parameter name.
format	Specifies the data type and length of *param_name.* Must be an OpenSQL data type supported by the CA-IDMS gateway. The syntax is: *datatype*

Parameter	Description
qualifiers	Optional parameter qualifiers: Only one of each type can be used per parameter.
	`[[not null]|[not default]|` `[with null]|[with default]|` `[default value]]` `|[[select_out]| [output]] |` `[[as name]|[is 'name']]`
	Valid defaults for data types are:
	date — CURRENT_DATE |CURRENT_TIMESTAMP |'quoted date string'
	char — 'quoted literal' | NULL | USER
	varchar — 'quoted literal' | NULL | USER
	float — numeric literal
	int — numeric literal
	System Defaults for data types (with default) are:
	Date — 0 date
	char — ' '
	varchar — ''
	float — 0.0
	int — 0
from '*source*'	Specifies the name or identification of the host DBMS or external procedure, if its name is different from *proc_name.*
	Enclose the value for *source* in single quotes.

For example:

```
register procedure usrprcc
      (p_category        char(10) not null,
      p_vehicle_id       char(03) not null
as import from 'usrprcc'
\p\g
register procedure "$DBAIIS6".selproc (i int,
    o1 int select_out default 0 is 'OUT1',
    o2 date select_out default current_timestamp is
                       'OUT2',
    o3 char(4) select_out is 'OUT3',
    o4 varchar(8) select_out is 'OUT4',
    o5 float select_out not null is 'OUT5')
as select_procedure        \p\g
```

Permissions

You can register a database procedure if you have been granted insert permission on the following system catalogs:

- IIGWPROCEDURES
- IIGWPROCPARAMS
- IIGWPROCDEFAULTS

The same rules are applicable for the remove procedure except that users must have been granted delete permission on the previously mentioned catalogs.

Executing Database Procedures

This section describes how to execute database procedures.

Preparing to Execute Procedures

Before you invoke a database procedure, you must create it on the host operating system. You must also register the procedure.

Execute Procedure Statement

To invoke a defined procedure, use the following syntax:

execute procedure *proc_name*
    (*param_name* = *param_value* {,*param_name* = *param_value*})
    into *return_status*)

Parameter	Description
proc_name	Procedure name
param_name	Input parameter name
param_value	A numeric or string literal or host variable that is compatible in type to the type of the parameter or the null constant
return_status	A host integer variable

Parameters are specified by name, not position, and are passed by value. Nullability and defaults follow standard EDBC semantics.

When the client issues the execute procedure statement, the gateway will load, initialize, and pass control to the specified procedure running under the gateway thread.

Select Procedure Statement

For select procedures, the execute does not actually invoke the procedure. It only sets up the parameters in the input and output SQLDAs.

The actual invocation is done using the following syntax:

```
select procname();
```

If you are using embedded SQL, the syntax is:

```
EXEC SQL select procname()
    into :var1, :var2,....
```

Only select loop processing is supported. Cursor processing or repeat queries are not allowed. The database procedure is effectively a repeated query.

Removing Database Procedures

To remove a procedure registration, use the following syntax:

remove procedure *proc_name*

where *proc_name* is the name of the procedure you want to drop.

The owner of the database procedure is the user who issued the create procedure statement.

Users can issue the remove procedure statement if they have been granted delete permission on the following system catalogs:

- IIGWPROCEDURES
- IIGWPROCPARAMS
- IIGWPROCDEFAULTS

Verifying Database Procedures

To test the functionality of database procedures, the stage2 job, IGWFVPM9, registers and executes sample database procedures. These database procedures are invoked by means of the EDBC/Terminal Monitor as a batch job. The following scripts are used as input to the Terminal Monitor run:

EDBC.V2R3.FILES.SQL(SASMTST)
EDBC.V2R3.FILES.SQL(SECHOPRC)

The source for these sample database procedures is described in the Implementing Database Procedures section.

Planning for EXCI Database Procedures

The External CICS Interface (EXCI), which is part of CICS4.1, has been implemented into the database procedure support of the gateway.

A sample Assembler database procedure (SASMECIT) has been provided. SASMECIT runs in conjunction with the EXCI sample server program DFH$AXCS. The EXCI server program accesses an external file that consists of 80 byte records. The records in the file are returned one at a time to SASMECIT, which sends them to the client as rows. Each 80 byte record has been broken down into 7 fields (columns). The column definitions are contained within the registration script for the SASMECIT database procedure (refer to SASMECIT in the FILES.SQL library).

To assemble SASMECIT, a CICS macro library (SDFHMAC) is required in order to pick up the DFHXCPLD macro.

To execute SASMECIT, a CICS load library (SDFHEXCI) is required in order for the gateway to pick up the EXCI interface programs.

Before attempting to test with SASMECIT, EXCI should be implemented and tested with the EXCI sample client program DFH$AXCC.

See the *CICS/ESA - External CICS Interface Manual* (SC33-1390-00) to do the EXCI install and test.

The gateway implementation uses the EXCI CALL Interface. Of the six EXCI calls within this interface, the gateway database procedure user just invokes the DPL_Request. All other calls are handled internally by the gateway and are not required within the database procedure. Within SASMECIT, a CALL_EXCI macro is used with a single parameter (a pointer to a DBPEXCI control block).

The DBPEXCI has a pointer to a communications area (COMMAREA). The COMMAREA is the data that is sent back and forth with the CICS transaction.

The database procedure only needs to deal with the DBPEXCI, the COMMAREA and the CALL_EXCI macro. The DBPEXCI will indicate a transaction id of EXCI and a program name of whatever program is to be invoked (DFH$AXCS in case of the sample server program). The COMMAREA will contain whatever the invoked program is expecting.

The logicals (for the EXCI sample) are:

- II_EXCI_APPL_ID = 'IICDCICS';
- II_EXCI_NET_NAME = 'BATCHCLI';

These logical are described in the “Logical Symbols and the IIPARM Clist” appendix.

Chapter

10 Optimizing and Troubleshooting

This chapter documents the several methods for improving the performance of applications that run against the gateway and debugging errors that may occur when you use the CA-IDMS gateway.

Optimizing Gateway Applications

This section is intended for application developers. It suggests several ways you can enhance the way your applications run against the gateway.

CA-IDMS Optimization

The gateway interacts with CA-IDMS as a standard CA-IDMS application. All CA-IDMS data accessed through the gateway remains completely under the control of CA-IDMS and is governed by CA-IDMS rules. As with any other CA-IDMS application, CA-IDMS determines data access permissions, optimum data access path, and data integrity and recovery.

Since CA-IDMS performs the query optimization, most query optimization can be done as you would any other CA-IDMS application. For example, you can utilize the CA-IDMS explain feature using the direct execute immediate statement. The explain command is a CA-IDMS-specific feature and is not OpenSQL, and therefore is not supported under OpenSQL. However, you can utilize this feature with the gateway using the direct execute immediate statement, which allows you to send non-OpenSQL statements directly to the host DBMS.

From the EDBC user interface, issue the following statement to invoke the CA-IDMS explain command for your query.

```
direct execute immediate
    explain statement 'SQL-statement'
```

CA-IDMS performs the explain on your query and stores the data access information in a CA-IDMS table ACCESS_PLAN with the key column, SECTION, set to 0. To access the results of the CA-IDMS explain, execute the following query:

select * from schema.access_plan

or you could use any EDBC user interface to access and format this data as you would any other table. For further information, see the *CA-IDMS SQL Reference Guide*.

Gateway Overhead

The gateway must translate queries and convert data types between OpenSQL and the CA-IDMS dialect of SQL. As a result, accessing CA-IDMS data through the gateway is somewhat slower than accessing the same data from a native CA-IDMS program. The gateway compares favorably with native applications doing fairly complex queries. The overhead is higher and the performance is slower when the gateway must process a large number of simple queries.

Gateway Query Handling

Different types of queries to the gateway require different amounts of processing. There are three categories of queries:

- For a direct execute immediate statement, which cannot return any data, the gateway:
  - Extracts the statement
  - Passes it on to CA-IDMS as a text string, without doing any translation, any syntax checking, or any further processing
- For an execute immediate statement, which also cannot return any data, the gateway:
  - Extracts the query
  - Translates the query from OpenSQL into the CA-IDMS dialect of SQL
  - Executes the query
- For a standard query that is going to return data, the gateway:
  - Extracts the query
  - Translates the query from OpenSQL into the CA-IDMS dialect of SQL
  - Converts the data types from OpenSQL to CA-IDMS SQL
  - Prepares the query by saving, encoding, and naming it

- Describes the query, defining a program descriptor to identify the data and data types for retrieval
- Opens a cursor
- Executes the query with an execute or a fetch statement

These differences in query processing affect how the gateway performs when it runs your application. For statements that do not return data, the gateway processes a direct execute immediate statement most rapidly. This type of query reduces gateway overhead to a minimum, but severely limits the portability of applications.

Other Ways to Increase Gateway Performance

You can enhance the gateway's performance by reducing the required communication between the EDBC user interface and the gateway. For example, a select loop buffers several rows of data from CA-IDMS in the gateway, then packages this data together and returns it as a single message. This minimizes the number of separate transactions that must occur between the gateway and the EDBC user interfaces.

The following features have different effects, but can all improve application performance:

- Using select loops
- Using gateway extensions to OpenSQL that provide a one-to-many mapping with CA-IDMS statements
- Using repeated queries

Using Select Loops

You can use select loops in an embedded OpenSQL program to retrieve multiple rows when no data will be updated. With a select loop, the gateway can declare and open a cursor, fetch all the rows, and close the cursor with a minimum of communication with the EDBC user interface. This is particularly useful when the users are remote and you want to minimize the traffic across the network.

Using Gateway Extensions to OpenSQL

Most queries through the gateway have a one-to-one mapping between the OpenSQL function and the corresponding function in CA-IDMS SQL. But certain extensions, such as create table as select, allow the user interface to issue a single OpenSQL command, which the gateway translates into multiple CA-IDMS-specific SQL commands. The CA-IDMS system executes these statements. The gateway then returns a message to the user interface indicating that the operation is accomplished.

Using Repeated Queries

If you are issuing a statement in an embedded OpenSQL program more than once, you can improve gateway performance significantly by using *repeated queries*. Repeated queries allow you to retain query definitions for the duration of a session. You can use program variables with a repeated query statement just as you would with a regular query statement. The variables are evaluated each time the statement is executed.

You can create repeated queries with the following OpenSQL statements:

- select (regular, singleton, or loop)
- insert
- update
- delete

To use a repeated query, preface the query statement with the key word repeated**.** For example:

```
exec sql repeated insert into table2 (col1, col2)
values (:ivar, :cvar)
```

CA-IDMS discards the query execution access when it encounters a commit, so be mindful of the effect that a commit statement has on repeated queries. The gateway does automatically prepare the query execution module a second time, if necessary, after a commit, without requiring any special action by the program.

Executing a Statement Dynamically

A statement that will be repeated can also be executed dynamically. Dynamic SQL for EDBC applications does not incur much additional overhead compared to non-dynamic statements using program variables.

Error Handling

This section discusses the how to handle run-time errors that can occur with the gateway.

CA-IDMS Error Reporting

When CA-IDMS reports an error, the gateway maps this error message to one of a series of generic error codes that are used across all the EDBC gateways. This mapping scheme enhances application portability.

Gateway Error Reporting

Some errors are detected by the gateway itself rather than by CA-IDMS. For example, if you mistype a command and the gateway cannot recognize the query, it provides an OpenSQL generic error message. This is displayed as a text message enclosed in parentheses. For example, if you type in:

```
sselect * from sales
```

the gateway returns the following generic error message:

```
(statement syntax error)
```

See the *OpenSQL Reference Guide* and the embedded SQL companion guides for the various host languages for further information on error handling.

You can access various types of error information using the techniques described in the following table:

Method	Returns
SQLCA.SQLCODE	OpenSQL Generic Error code.
SQLCA.SQLERRD(0)	CA-IDMS SQL SQLCODE: an array of integers. The first entry in that array is the CA-IDMS-specific error code.
SQLCA.SQLERRM	First 70 characters of the CA-IDMS error message text. This also contains the variables if the message comes from CA-IDMS.
whenever SQLERROR call SQLPRINT	Entire CA-IDMS error message text.

See the *OpenSQL Reference Guide* for details on SQLCA (SQL Communications Area).

Debugging the Gateway

The gateway is primarily responsible for translating between OpenSQL and the CA-IDMS dialect of SQL. Most errors in gateway operation can be classified as one of the following types:

- Syntax errors
- Incorrect return codes
- Incorrect semantics

- Data type errors

The following sections describe these errors in greater detail. They also offer suggestions for isolating the cause of the error and for correcting it.

Syntax Errors

In this type of error, the gateway returns an error message when a query is run against CA-IDMS. When the same query is submitted against a local database, no error message is returned. There are several possible causes for syntax errors:

- The query is valid EDBC SQL but not valid OpenSQL. Consult the *OpenSQL Reference Guide* for the correct statement syntax.
- The query is valid CA-IDMS SQL but not valid OpenSQL. Use the direct execute immediate statement to pass CA-IDMS SQL statements through the gateway.
- The gateway routine does not pass the correct CA-IDMS SQL statements or data to CA-IDMS. Report the problem to your site contact for Computer Associates' Technical Support.
- The gateway passes the appropriate CA-IDMS SQL and data to CA-IDMS, but CA-IDMS does not handle these correctly. Check your CA-IDMS documentation for possible programming errors.
- You are running a version of CA-IDMS that is not compatible with the gateway. For software version requirements, see the Software Requirements section of the "Preparing for Installation" chapter.

Incorrect Return Codes

This type of error occurs when a query is run against CA-IDMS and does not return the expected error message. When the same query is run against a local EDBC database, it returns the correct message.

To correct this error, the CA-IDMS error message must be mapped to the correct EDBC Generic Error Message. Report the problem to your site contact for Computer Associates' Technical Support.

Incorrect Semantics

Sometimes a query returns the expected results when run against an EDBC database, but fails to do so when run against CA-IDMS. If the gateway is passing the correct SQL statements and data to CA-IDMS this type of error indicates an incompatibility between the two SQL dialects. Report the problem to your site contact for Computer Associates' Technical Support.

Data Type Errors

A data type error occurs when the gateway cannot translate CA-IDMS data into the corresponding gateway data type. This occurs most often when EDBC applications pass hard-coded dates to the gateway as character strings.

Gateway Traces

When you need to use a trace to determine the cause of a problem, you can use one of the following two methods to initiate a trace:

- By means of logical symbols in the IIPARM startup member.

  This method affects all gateway connections so it should not be used during high use time periods (an alternate server, as described in the "Maintaining the Gateway" chapter, could be set up just for tracing if there are no good time periods for this overall tracing).
  There are two logical symbols that can be used to initiate an overall trace.

  - GW_TRACE_INIT = 'IBOBEXRSERDATR';

    This logical symbol starts tracing at the earliest possible moment during the client connection. Any problems going to CA-IDMS can be uncovered by using this logical. (The two character values are discussed in the following method.)

  - ING_GW_SET = 'SET TRACE IB,OB,EX,RS,ER,DA,TR LOCAL';
    ING_GW_SET = 'SET TRACE ALL LOCAL';

    This logical symbol starts tracing after the CA-IDMS connection has been established. (The two character values are discussed in the following method.)

- The client may send in the set trace command to start an individual trace.

  This method allows a set notrace command to be used to turn off the individual trace. The format of the set trace command is:

  SET TRACE {*ID*[,*ID...*] | *ALL*} [*REMOTE* | *LOCAL* | *BOTH*]}

  where REMOTE (send output back to client) is the default.

  where Trace point IDentifiers are:

  - IB - Inbound messages
  - OB - Outbound messages
  - EX - Executing SQL
  - RS - Results of SQL
  - ER - Error conditions

- DA - Data (SQLDA) contents
- TR - Trace points
- ME - Memory allocate/free
- HO - Host information (This option generates a large amount of output.)

Resetting the OS/390 Subsystem

At times, it may be necessary to reset the OS/390 subsystem for the EDBC server. For instance, if the server abends or stops abnormally, it is possible that the next time it is started that it will not be able to come up and will end with a return code of 20.

When this occurs, the OS/390 subsystem must be reset.

Chapter 11 Using Table Procedures

The CA-IDMS gateway provides support for table procedures to allow remote users to invoke user-written procedures on the host operating system.

Your primary source of information about table procedures is the documentation for the appropriate platform. This chapter does not summarize that information. Instead, it addresses the aspects of working with table procedures that are particular to CA-IDMS. This chapter:

- Summarizes the input parameters provided to table procedures
- Discusses accessing table procedures
- Describes the table procedure functions
- Outlines the coding conventions for table procedures
- Discusses special considerations for table procedures

Table Procedures

A table procedure is a user-written program that allows any data accessible through CA-IDMS to be viewed and processed as a table. The parameters passed to and from the program are treated as the columns of a table which, once defined to CA-IDMS as a table procedure, can be operated on using SQL DML statements. The specifics of how the database is accessed in handling these requests is hidden within the table procedure. The SQL referred to in this chapter is the CA-IDMS SQL.

When to Use Table Procedures

Table procedures are used to process non-SQL defined data in a relational way although the data does not conform to the rules established for such access. For example, table procedures can be used to:

- Provide full update capability on member records that do not contain embedded foreign keys
- Access data with multiple definitions

- Access data that does not conform to the data type defined in the non-SQL schema
- Translate special data values into null values
- Make the processing of complex structures, such as bill-of-materials, easier for an end user.

  Since the details of access to the underlying records or tables is encapsulated within the procedure, less knowledge is required on behalf of the user to process the data.
- Allow access to all segments of a segmented database within a single SQL transaction.

  Since a table procedure can open more than one rununit or SQL session simultaneously, it can access the appropriate segment based on the value of an input parameter. If no appropriate segment key is available, it can serially access each segment.
- Access remote data.

  This enables a single SQL transaction to access data distributed across different nodes within a CA-IDMS network while hiding the knowledge of the location of the data within the procedure itself.

Defining and Using Table Procedures

The following sections describe how to define and use table procedures.

Defining a Table Procedure

Use the create table procedure statement to define a table procedure as shown in the following example:

```
create table procedure emp.org
 (top_key            unsigned numeric(4),
     level            smallint,
     mgr_id           unsigned numeric(4),
     mgr_lname        char(25),
     emp_id           unsigned numeric(4),
     emp_lname        char(25),
     start_date       char(10),
     structure_code   char(2))
     external name    procorgu;
```

The table procedure, org, is named and associated with the schema, emp. The program to be called to service a DML request against the procedure is specified in the external name parameter as procorgu.

The parameters to be passed to and from the procedure are then listed. Each parameter definition should at least contain a name and a data type.

For more information on defining table procedures, see the *SQL Reference Guide*.

Accessing Table Procedures

Use SQL DML statement to access table procedures in the same way base tables and views are accessed. Table procedures can be referenced any place a table reference is permitted. Whether or not an SQL operation is supported by a specific table procedure is dependent on the user-written program. It may, for example, support only retrieval operations and disallow insert, update, and delete by returning an error if such an operation is attempted.

Access to a table procedure is controlled just as if it were a table. The grant and revoke statements on a resource type of table are used to give and remove select, insert, update, delete or define privileges on a table procedure.

Procedure Parameters

Table procedure parameters are treated like table columns. They can be specified within the column list of a select or insert statement, the set clause of an update statement, the order by clause of a select statement or the search criteria of a where clause. In addition, input parameter values may be specified within the procedure reference itself.

Column List, Set, and Order By References

Parameters referenced in the:

- Column list of a select statement specify the columns that will be returned to the invoking application
- Column list of an insert statement specify the columns whose values are supplied in the subsequent values clause or query-specification
- Set clause of an update statement specify the columns which will be assigned new values during the update operation
- Order by clause of a select statement determine the order in which the result rows of the procedure are returned to the requesting application

Where Clause References

Where clause references to parameters filter a procedure’s output. Each time a procedure returns a set of output values, they are evaluated against the selection criteria specified in the where clause and non-conforming rows are ignored.

Also, where clause parameter references can pass input values to the procedure. For example, selection criteria of the form:

```
<Parameter> = <Value>
```

results in <Value> being passed to the procedure. Other types of selection criteria, such as in predicates or comparison predicates with > or < operators have no effect on the value of the parameters passed to the procedure.

Specifically, a reference to a parameter in a where clause results in an input value being passed to the procedure only if the following occurs:

- It appears within an equality test
- The equality test is not combined with other predicates in the where clause through the use of the or operator
- The equality test is not preceded by the not operator

The following table illustrates these points:

Where Clause	**Parameter Value**
P1	P2
P1 = 1	1 -null-
P1 < 1	-null- -null-
P1 = C1	C1 -null
P1 = 2 AND P2 = 3	2 3
P1=2 AND P2>3	2 -null-
P1 = 2 OR P2 = 3	-null- -null-
P1 IN (2, 3, 8)	-null- -null-

Parameters in Procedure References

Input parameter values can also be specified within the procedure reference itself. They may be specified on any reference to a procedure except within an insert statement.

Parameter values supplied on a procedure reference may be specified either by their position or as keyword/value pairs. They may also be combined with where clause references to form the set of values that will be passed to the procedure.

All of the following are valid ways in which to specify input parameter values.

```
Select * from emp.org (mgr_id = 7, emp_id = 127)
select * from emp.org (cast(null as num(4,0)), 7, 127)
select * from emp.org (mgr_id = 7) where emp_id = 127
```

Differences in Parameter Specification

There is a difference between parameter values specified through a where clause and those specified within the procedure reference. Parameter values specified within the procedure reference are not used to filter the output from the procedure as is the case for those specified within the where clause. Parameter values specified within the procedure reference affect only the input to the procedure and not the output from the procedure.

Coding Table Procedures

The program associated with a table procedure may be written in COBOL, PL/I, or Assembler. When called, the program is passed a fixed parameter list consisting of the parameters specified on the procedure definition as well as additional parameters used for communication between CA-IDMS and the procedure.

Whenever a reference to a table procedure is made, CA-IDMS calls the program associated with the procedure to service the request. Part of the information passed to the procedure is an indication of the type of action that the procedure is to perform, such as return the next result row or update the current row. The procedure responds by performing the requested action or returning an error.

Transaction and session management are performed automatically by CA-IDMS in response to requests issued by the originating application. Changes to the database made by a procedure are committed or rolled out together with other changes made within the SQL transaction. No special action is required of the procedure in order to ensure this occurs.

Calling Arguments

Each time a procedure is called, it is passed the following sets of arguments:

- one argument for each of the parameters specified on the procedure definition, passed in the order in which the parameters were declared
- one argument for each null indicator associated with a parameter specified in the procedure definition, passed in the order in which the parameters were declared
- a set of common arguments used for communications between CA-IDMS and the procedure

The first two sets of arguments described above will vary from one procedure to another. They are used to pass selection criteria and insert/update values to the procedure and result values from the procedure.

The last set of arguments is the same for all procedures as shown in the following table:

Argument	Contents
Result Indicator (fullword)	Not used
sqlstate (**char**(5))	Status code returned by the procedure as follows: 00000 - indicates success 01Hxx - indicates a warning 02000 - indicates no more rows 38xxx - indicates an error
Procedure Name (**char** (18))	Name of the procedure.
Program Name (**char** (8))	Name of the program.
Message Text (**char** (80))	Message text returned by the procedure and displayed by CA-IDMS in the event of an error or warning.
SQL Command Code (fullword)	Code indicating the type of SQL request for which the procedure is being called.
SQL Operation Code (fullword)	Code indicating the type of request being made of the procedure. The Procedure Calls section contains a list of valid operation codes.
Instance Identifier (fullword)	A unique value identifying the scan on which the procedure is to operate.
Procedure Work Area (user-defined)	A user-defined storage area maintained across calls to the procedure.

Procedure Calls

Part of the information passed to the procedure is the type of request being made. This information is conveyed in two parameters. The first contains a code indicating the type of SQL statement for which the request is being issued (for example, insert, or open); the second is an internal operation code indicating the type of action expected of the procedure.

The following operation codes are possible as shown in the following table:

Operation Code	Meaning
Open Scan (value 12)	Requests that the procedure prepare itself for returning a set of result rows. Selection criteria specified in the where clause or in the procedure reference may be passed as arguments to the procedure.
Next Row (value 16)	Requests that the procedure return the next result row for the indicated scan. Next Row requests are repeated in order to return all of the result rows for a scan. The procedure can set an sqlstate value indicating that all rows have been returned.
Close Scan (value 20)	Informs the procedure that no further next row requests will be issued for the scan. The procedure may free resources in response to this request.
Update Row (value 40)	Requests that the procedure update the current row of the indicated scan using the values of the passed parameters as the update values. Update row requests are issued in response to either searched or positioned update statements.
Delete Row (value 36)	Requests that the procedure delete the current row of the indicated scan. Delete row requests are issued in response to either searched or positioned delete statements.
Insert Row (value 32)	Requests that the procedure insert a row into the database using the values of the passed parameters as the insert values.
Suspend Scan (value 24)	Informs the procedure that the SQL session is being suspended. The procedure may release resources in response to this request.
Resume Scan (value 28)	Informs the procedure that the indicated scan is being resumed following a suspend. The procedure may re-establish its state if necessary.

Both select statements and open, fetch, or close cursor requests will result in the following set of calls to the procedure:

```
open scan
    next row (1 to n times)
close scan
```

A searched update statement will result in the following:

```
open scan
    next row   \ (1 to n times)
    update row /
close scan
```

A positioned update statement associated with an open cursor will have a similar calling sequence except that an update row request may not be issued after every next row request.

Searched and positioned delete statements result in similar calling sequences to those for searched and positioned update statements.

Insert statements result in a single call to the procedure for each row to be inserted.

Parameter Arguments

On entry to the procedure, the value of the arguments corresponding to the parameters defined on the create procedure statement vary depending on the type of operation being performed, as follows:

On an Open Scan request, non-null parameters contain one of the following:

- The selection criteria specified in the where clause
- The parameter values specified on the procedure reference
- The datatype-specific default value if with default was specified in the procedure definition

All other parameters contain nulls (for example, the null indicator for the parameter is negative).

On an update row request, the parameters contain the values returned from the previous Next Row request, overlaid with the values specified in the set clause of the update statement.

On an insert row request, the parameters contain the values specified in the values clause of the insert statement or the values returned by the select associated with the insert statement. Unspecified values are either null or contain the parameter’s default value.

On other types of requests, the contents of the parameters are undefined on entry.

On exit from a next row request, the procedure is expected either to have set the value of the parameter arguments and their indicators appropriately or to have set an SQLSTATE value indicating no-more-rows. If an indicator parameter is set to -1, the value of the corresponding parameter is ignored by
CA-IDMS.

Instance Identifier

On every call issued to a procedure, a parameter is passed identifying the scan to which the request is being directed.

In the case of insert, this has no meaning; however in all other cases (select, update, delete, and cursored operations the instance id is used to distinguish one execution thread from another.

If an application program is written to have two simultaneously open cursors both of which reference the same procedure, the procedure must be able to distinguish Next Row calls associated with one cursor from Next Row calls for the other. The instance identifier is used for this purpose. Each cursor results in a separately opened scan and each scan is assigned a unique instance identifier. The procedure is responsible for maintaining enough information for each scan (such as database keys) to enable it to return the next row of the scan.

Procedure Work Area

Another parameter passed on each call to a procedure is a work area in which the procedure may save information it wishes to preserve from one call to another. Information cannot be saved within program variables (such as working storage) because the program is invoked using a #LINK operation, causing the program variables to be re-initialized on every call. The type of information which may need to be preserved across calls are the:

- Subschema control block for a rununit or the session identifier of an SQL session
- Database position information
- Input parameter values used as selection criteria

A procedure work area is allocated by CA-IDMS when the first call to a procedure is made. A different work area may be allocated each time a new scan is initiated, or the same work area may be shared for all scans with which the procedure is associated. The size of the work area and whether or not it is shared across scans is specified as part of the procedure definition.

A procedure which updates the database should do so through only one rununit or SQL transaction to avoid deadlocking against itself. So typically an update procedure will use a shared work area in order to allow access the same subschema control or SQL session identifier. A retrieval-only procedure may instead use a separate work area for each scan, opening the rununit or SQL session on the Open Scan request and terminating it on the Close Scan request.

Special Considerations

The following sections describe special considerations you should be aware of when using table procedures.

Environment Independence

Since it is likely that a procedure will execute both within a batch address space because of local mode access and within the DC/UCF address space, it should be written to be independent of the runtime environment.

If a procedure will be executed in local mode (for example through IDMSBCF), then it must either limit itself to database requests only or be written in Assembler (or use an Assembler subroutine for DC/UCF requests)

Even if it is written in Assembler, many DC services such as Queue, Print, and terminal I/O or not supported in local mode and should be avoided. Scratch and storage requests issued from an Assembler program are supported in local mode.

If a procedure will be executed within DC/UCF, it should not contain statements that would interfere with or are prohibited in the that environment. For example DISPLAY statements in COBOL and getmain requests in Assembler should be avoided. Follow the rules specified in the appropriate DML referenced guide when coding your procedure.

Transaction Management

Rununits and SQL transactions opened within a procedure are automatically managed as part of the invoking SQL session's transaction. This means that if a commit work is issued by the invoking application, all subordinate transactions opened by procedures are also committed. Similarly, if the invoking SQL session is rolled out, all subordinate transactions are also rolled out.

Although a procedure is free to terminate its own transactions independently from the invoking SQL session, it should do so only if it made no changes to the database.

When the invoking SQL transaction is terminated, either through a commit or rollback operation, it has the following affect on procedures:

- All open scans are closed
- All database transactions (SQL or non-SQL) started by the procedure are either committed or rolled out and the corresponding sessions are terminated
- All procedure work areas are freed

When the invoking SQL transaction is committed through a commit continue operation, the only affect on procedures is that database changes made by the procedure are committed.

Suspend/Resume

If an SQL session which invokes a procedure is suspended, the procedure is also suspended. This means the following:

- Rununits and SQL sessions started by the procedure are suspended

- Database changes made by the procedure (whether through SQL or native DML) are neither committed nor rolled out; instead the records remain locked and the changes will either be committed or rolled out when the invoking SQL session is committed or rolled out
- The work area associated with the procedure is saved

In most cases, no special action is required of the procedure during a suspend operation. The procedure will be called once for each open scan that exists at the time the session is suspended. If the procedure has acquired some temporary storage, it may need to save its contents somewhere else (such as in kept storage or in a scratch area) that will be preserved across a pseudo-converse; otherwise, the procedure may ignore suspend scan requests.

Similarly, the procedure may ignore resume scan requests unless it needs to restore a temporary storage area. The first request to a procedure after a resume will be a Resume Scan if the request involves a scan that was previously suspended. If the request does not involve a previously suspended scan, the procedure may not even be aware that a suspend and resume has occurred. The contents of the procedure's work area will be the same as it was before the suspend and any rununits or SQL sessions previously started by the procedure will automatically be resumed on the next database request.

Error Handling

The procedure is provided with two arguments to signal an exception condition back to CA-IDMS. These arguments consist of a 5-character code known as SQLSTATE and an 80-byte message area. The valid SQLSTATE values and their meanings are shown in the following table:

SQLSTATE Value	Meaning
00000	Request was successful
01Hxx	Request was successful but the procedure generated a warning message
02000	No more rows to be returned
38xxx	The procedure has detected an error during processing

CA-IDMS examines the SQLSTATE value to determine whether the operation was successful or not. In the case of either an error or a warning, it embeds the message text returned by the procedure in a standard DB message and returns it to the calling application through the message area of the SQLCA. It also translates the SQLSTATE value into an SQLCODE value as follows:

SQLSTATE	SQLCODE
00000	0

01Hxx	1
02000	100
38xxx	-4

In the event an error is raised by the procedure, CA-IDMS automatically rolls out all database changes made by the procedure while processing the SQL statement that caused the procedure to be invoked. For example, if the invoking SQL statement was a searched update and ten rows had been updated before the error was detected, changes to all ten rows are rolled out automatically. Database changes made prior to the execution of the searched update statement are not rolled out.

Datetime Parameters

If a table procedure is defined to have a parameter whose data type is date, time, or timestamp, the values passed to and from the procedure are in the 8-byte internal datetime format. This means that in order to interpret incoming parameters, the procedure must first convert them to external format using either the IDMSIN01 subroutine or the #XTRA macro. Similarly, before returning a datetime parameter value, the procedure must convert the external format to the internal format again using either IDMSIN01 or #XTRA.

To avoid this, date/time parameters may be defined using a character data type which may then be converted to a date, time, or timestamp using the cast function. However, this method relies on the invoking application or end-user for specifying the cast operation and on the procedure for doing the datetime validation on update and insert values.

Transaction Mode

The transaction mode of an SQL session which invokes a procedure is propagated to the subordinate transactions started by the procedure. This means that if the procedure starts an SQL session, its default transaction mode will be the same as the transaction mode associated with the invoking transaction. If the procedure instead binds a rununit and the transaction mode of the invoking SQL transaction is read only, all update ready modes will automatically be converted to shared retrieval. The net result is that a table procedure invoked by a transaction in a read only state, will not be able to update the database.

Appendix

A Logical Symbols and IIPARM Clist

EDBC server and gateway behavior is controlled by logical symbols that are specified when EDBC is started. The IIPARM DD statement determines which logical symbols are used. In the batch environment (batch job or started task), IIPARM can point to a sequential data set or a member of a PDS.

In the foreground (TSO), the IIPARM clist supplied in the EDBC.V2R3.FILES.CLIST data set is used to specify the member of a PDS containing the logical symbols to be used for the session. The EDBC.V2R3.FILES.IIPARM data set contains default logical symbol members that are created from user stage1 macro specifications.

Logical Symbol Library Organization

The Logical Symbol library has the default name EDBC.V2R3.FILES.IIPARM. This library contains members that are used for different purposes. These members are:

- ISVRxxxx member

  Logical symbols to initialize or access an EDBC server. This member is created from user stage1 macro specifications.

- SABExxxx member

  The CA-IDMS gateway can be invoked independently of the gateway server. In other words, the CA-IDMS gateway can be run as a batch job or under TSO, even though the gateway server is not started. This mode of operation is called Stand Alone Back End (SABE) and is determined by the logical ING_MODE. Specifying ING_MODE=NOSERVER; will enable the CA-IDMS gateway to be invoked independently of the gateway server. Specifying ING_MODE=SERVER; indicates that the batch job or TSO user will interface to the CA-IDMS gateway through the gateway server and thus requires that the gateway server be started. In addition to the ING_MODE logical, several other logicals are required to run in Stand Alone Back End mode. These logicals are generated and stored in library EDBC.V2R3.FILES.IIPARM as member SABEDEMO by the stage2 installation process.

- Uxxxxxxx member

  These members contain the same type of logical symbols as those in the SABExxxx member, with which they can be interchanged.

  The advantage of this set of members is that you can customize up to seven characters of the member name, as opposed to only four characters of the SABExxxx member set.

Logical Name Format

The gateway uses logical names to configure many operational parameters. These logicals have the following syntax:

```
LOGICAL_NAME = 'value';
```

You can customize logicals by editing the text that appears to the right of the equals (=) sign. Each record in a logical file may be:

- Blank
- An assignment, using the syntax:

  ```
  LOGICAL_NAME = 'value';
  ```

  where the value is expressed as a string or a number, depending on the logical name. Enclose string values in single or double quotes. Each logical name assignment must end with a semicolon.
- A comment, delineated by /* and */
- An assignment followed by a comment

Detailed Descriptions of Logical Symbols

This section describes the logical symbols you can configure. A logical symbol can be specified more than once. If this happens, the system uses the last occurrence of that logical symbol. Logical symbols are grouped into several members. This section describes the symbols that are contained in each of the following members:

- ISVREDBC: Server logical symbols
- SABExxxx: Stand-alone back end logical symbols

ISVREDBC Logical Symbol Members

These logical symbols are used to initialize the gateway server and to provide access to that server from a remote client, a batch, or a TSO address space. The IIPARM Clist also uses these logical symbols in conjunction with symbols in the DBNMEDBC and SABExxxx members.

IDMS_CI_OPTION

The IDMS_CI_OPTION logical symbol enables you to append CA-IDMS-specific parameters to the create index statement. The form of this option is:

```
IDMS_CI_OPTION = 'IN segment.areaname '
```

IDMS_CT_OPTION

The IDMS_CT_OPTION logical symbol enables you to append CA-IDMS-specific parameters to the create table statement. The form of this option is:

```
IDMS_CT_OPTION = 'IN segment.areaname '
```

IDMS_DECIMAL_AS_CHAR

IDMS_DECIMAL_AS_CHAR can be used to direct the gateway to return decimal data as character format instead of float format. (The gateway does not directly support decimal.) For example:

```
IDMS_DECIMAL_AS_CHAR = 'YES';
```

This example will return decimal/numeric fields in character format with a decimal point if the scale of the field is non-zero. A minus sign will precede the field if it was negative. The overall length of the data returned will be equal to the precision of the field plus 2 (sign + decimal (even if none)).

Select DBMSINFO('decimal_as_char') will return an ON/OFF value to show the state of the logical.

Set decimal_as_char on/off can be used to have the gateway alter the iimapping table to the appropriate values. If in the off condition, iimapping will indicate that the returned fields are float. The on condition will cause iimapping to be set to return char for these fields. (IIMAPPING is used when IICOLUMNS is accessed for things such as HELP TABLE processing.)

This set needs only be used once to alter the IIMAPPING table.

The user should be consistent in the use of this logical and the values in IIMAPPING. A choice should be made as to which way to have decimal data returned. If the choice is CHAR, then set the logical as above and issue the SET. Otherwise, the default of FLOAT, which is consistent with previous gateway releases, will be used.

IDMS_DEFAULT_DBNAME

This logical symbol specifies the default dbname to be connected to during the gateway installation. The form of this option is:

```
IDMS_DEFAULT_DBNAME = 'EDCSQL';
```

IDMS_HOT_CONNECT

This logical symbol specifies whether a hot connection is to be established with CA-IDMS during EDBC server initialization. The valid values are YES and NO. For example:

```
IDMS_HOT_CONNECT = 'YES';
```

Hot connections to CA-IDMS can also be started by the ACTivate operator command.

GW_TRACE_INIT

The GW_TRACE_INIT logical symbol specifies which gateway trace items are to be written to the IIGWTR output file. For example:

```
GW_TRACE_INIT='IBOBEXRSER';
```

where the values are:

IB—inbound messages
OB—outbound messages
EX—SQL executions
RS—SQL results
ER—error conditions
ME—memory manipulations
DA—SQLDA data items
TR—trace results

This logical can produce lots of output. You can use the Terminal Monitor command as follows to control tracing:

```
set trace all|list of items
```

II_DATE_CENTURY_BOUNDARY

In the gateway, the year defaults to the current year. In formats that include delimiters (such as forward slashes or dashes), you can specify the last two digits in the year; the first two digits default to the current century (1900). For example, if you enter 03/21/93 using the format *mm/dd/yyyy*, the gateway assumes that you are referring to March 21, 1993. This behavior forces the user to specify all 4 digits of the year when dealing with dates in the next century.

The II_DATE_CENTURY_BOUNDARY logical symbol allows you to change this default. It contains an integer that has a value between 0 and 100 inclusive. If this logical symbol is set, and if the setting is in the valid range, then the century will be determined by the following calculation:

```
if (current_year < boundary)
    if (input_year < boundary)
output_year = input_year + current_century
    else
output_year = input_year + previous_century
else
    if (input_year < boundary)
output_year = input_year + next_century
    else
output_year = input_year + current_century
```

Thus:

II-DATE-CENTURY-BOUNDARY= ‘93’;

Then an input date of 03/31/93 will be treated as March 21, 1993. However, an input date of 03/21/03 will be treated as March 21, 2003.

II_DBMS_LOG

This specifies the location of the DBMS error log. The value can be either a ddname or a dsname.

- ddname

  If a ddname is given as the value, it is in the form DD:*ddname* where *ddname* is the value of a data definition (DD) statement that was included in the gateway server JCL.

- dsname

  If a dsname is given as a value, it is in the form *dataset.name* where *dataset.name* is the fully qualified name of an existing sequential data set. This data set will be dynamically allocated and used to log database management system errors. An example is:

```
II_DBMS_LOG = 'DD:IIERLOG';
```

II_DBTMPL

This specifies the name of the non-relational gateway database template directory. This symbol must be entered exactly as shown:

```
II_DBTMPLT = 'ING_AREA:DB.iicore';
```

II_FILES

This specifies the OS/390 data set prefix of the various gateway installation data sets. If this symbol is not defined, then the gateway will concatenate the value of the SYS_INGRES logical symbol with the character string .FILES to create the data set prefix. An example is:

```
II_FILES = 'EDBC.V2R3.FILES';
```

II_FORCE_TMOUTINT

This specifies the number of minutes of inactivity, following an attempted user disconnect, an operator inactivate, or an inactivate user timeout, that the system will wait before forcing the user connection down if it still exists. This is similar to the inactive user timeout facility but is a second-stage more severe form of a “kill.”

The default is 5 minutes. An example is:

```
II_FORCE_TMOUTINT = 5;
```

II_FORMFILE

This specifies a front-end parameter that defines the location of cached forms files. An example is:

```
II_FORMFILE = 'EDBC.V2R3.FILES.RTIFORMS.FNX';
```

II_GCC_ID

This specifies the OS/390 subsystem name that is used by the gateway address space. This value must match the value of the II_GCN_ID symbol. An example is:

```
II_GCC_ID = EDBC;
```

II_GCCI1_LOG

This specifies the location of the gateway communications server error log. The value given can be either a ddname or a dsname.

- ddname

  If a ddname is given as a value, it will be of the form DD:*ddname* where *ddname* is the value of a data definition (DD) statement that was included in the EDBC server JCL.

- dsname

  If a dsname is given as a value, it will be of the form *dataset.name* where *dataset.name* is the fully qualified name of an existing sequential data set. This data set will be dynamically allocated and used to log communication server errors.

  If the II_INSTALLATION symbol has a value other than II, then this logical symbol will have the characters II replaced by the new specification. An example is:

  ```
  II_GCCI1_LOG = 'DD: IIERLOG';
  ```

II_GCN_ID

This specifies the OS/390 subsystem name that is used by the gateway name server. This value must match the value of the II_GCC_ID symbol. An example is:

```
II_GCN_ID = EDBC;
```

II_GCNI1_LOG

This specifies the location of the gateway name server error log. The value given can be either a ddname or a dsname.

- ddname

  If a ddname is given as a value, it will be of the form DD: *ddname* where *ddname* is the value of a data definition statement (DD) that was included in the EDBC server JCL.

- dsname

  If a dsname is given as a value, it will be of the form *dataset.name* where *dataset.name* is the fully qualified name of an existing sequential data set.

  This data set will be dynamically allocated and used to log communication server errors. If the II_INSTALLATION symbol has a value other than II, then this logical symbol will have the characters II replaced by the new specification. An example is:

  ```
  II_GCNI1_LOG = 'DD: IIERLOG';
  ```

II_GCNI1_LCL_VNODE

This specifies the virtual node name of the gateway Comm server. It is used by the name server to determine whether a remote connection or local connection is desired. If it is not present, the name server will always attempt a remote connection. If it is present, the name server compares the requested virtual node to this value. If there is a match then the gateway is started locally in the TSO or batch address space. If the II_INSTALLATION symbol specifies a value other than I1, then the symbol's name must be changed accordingly. An example is:

```
II_GCNI1_LCL_VNODE = EDBC;
```

II_GCNI1_SVR_TYPE

This specifies the default server class that will be used during a connection process if a server class is not given in the connect statement. The valid values are:

- DCOM—specifies the default server class, the CA-Datacom/DB gateway.
- DB2—specifies the default server class, the DB2 gateway.
- IDMS—specifies the default server class, the CA-IDMS gateway.
- IMS—specifies the default server class, the IMS gateway.
- VSAM—specifies the default server class, the VSAM gateway.
- VANT—specifies the default server class, the Vantage gateway.

II_GENERIC_ERROR

The gateway maps CA-IDMS-specific errors to a series of generic errors. This mapping scheme makes application-error-handling portable across different gateway products and platforms.

The valid values are YES and NO. If the logical symbol is defined as YES, when the gateway detects an error in the SQL statement execution, it returns the following:

- OpenSQL Generic Error Code
- CA-IDMS Error Code

If the logical is defined as NO, the Data Manager returns the standard error messages. An example is:

```
II_GENERIC_ERROR = YES;
```

II_HELPDIR

This specifies the data set that contains the Terminal Monitor help files. The help files are members of the data set with the default name EDBC.V2R3.FILES.HLP, which was part of the installation tape. An example is:

```
II_HELPDIR = "PDS:'EDBC.V2R3.FILES.HLP'";
```

II_IDMSLU62_ACB

This logical symbol supplies the VTAM ACB name to use for the CA-IDMS LU62 PIPE.

For example:

```
II_IDMSLU62_ACB = 'IDMSSY11';
```

This logical is set by default by the IGWFIDMS ACB= parameter.

II_IDMSLU62_ LOGMODE

This logical symbol supplies the VTAM LOGMODE entry to use for the CA-IDMS LU62 PIPE.

For example:

```
II_IDMSLU62_LOGMODE = 'APPC02A';
```

This logical is set by default by the IGWFIDMS DLOGMODE= parameter.

II_IDMSLU62_TASKID

This logical symbol supplies the task code to be passed by the gateway to CA-IDMS for invoking the gateway transaction.

For example:

```
II_IDMSLU62_TASKID = 'RSPD';
```

This logical is set by default by the IGWFIDMS TASKID= parameter.

II_INACTV_TMOUTINT

This specifies the number of minutes of inactivity after which user threads are timed out. User threads that may be "hung" as a result of users breaking out of front-end applications (for example, pressing PA1 in TSO) are automatically released after the specified timeout. Security is enhanced because unattended sessions are automatically logged out. Gateway threads and resources are freed up.

If the parameter is not specified, or if it is set to zero, the timeout facility is not activated—that is, no automatic timeouts will occur. If you specify the parameter but do not assign it a value, it defaults to 0. An example is:

```
II_INACTV_TMOUTINT = 0;
```

II_INSTALLATION

This specifies the 2-character code that is the gateway installation ID. The default value for this code is I1. It must be unique for each gateway server address space.

This 2-character code appears in the following logical symbols:

```
II_GCCii_LOG
II_GCNii_LCL_VNODE
II_GCCii_SVR_TYPE
```

where ii is the value specified in the II_INSTALLATION symbol.

This installation code can also appear in several logical symbols that are not normally used:

```
II_GCCii_LOGLVL
II_GCCii_ERRLVL
```

This installation code, along with the value of the II_FILES logical symbol, is used to create the OS/390 data set names that comprise the name server database.

These files are:

```
EDBC.V2R3.FILES.NAME.IICOMSVR.ii
EDBC.V2R3.FILES.NAME.IIDB2.ii
EDBC.V2R3.FILES.NAME.IIIMS.ii
EDBC.V2R3.FILES.NAME.IIVSAM.ii
EDBC.V2R3.FILES.NAME.IIEDBC.ii
EDBC.V2R3.FILES.NAME.IILOGIN.ii
EDBC.V2R3.FILES.NAME.IINODE.ii
EDBC.V2R3.FILES.NAME.IIVSAM.ii
```

where ii is the value of the II_INSTALLATION logical symbol.

An example of the default entry is:

```
II_INSTALLATION = I1;
```

II_NET_LMOD

This logical symbol specifies the name of the IIPSERV (protocol server) load module to use for EDBC initialization. The default value is IIPSERV.

```
II_NET_LMOD = 'IIPSERV';
```

II_NO_ENQ_SUBSYS

This specifies that the gateway be invoked within the EDBC address space. It must be specified as follows:

```
II_NO_ENQ_SUBSYS = YES;
```

II_PIPE_IDMSLU62

This logical symbol specifies whether the CA-IDMS LU62 PIPE is to be activated during Net server initialization. The valid initialization values are YES and NO.

For example:

```
II_PIPE_IDMSLU62 = 'YES';
```

The CA-IDMS LU62 PIPE can also be started by the ACTivate operator command.

II_PROTOCOL_CCI

The valid values are YES and NO. If the logical number is defined as YES, it specifies that the installation is using the CCI protocol for OS/390 to OS/390 communications. An example is:

```
II_PROTOCOL_CCI = YES;
```

II_PROTOCOL_ RESTART_COUNT	The number of times the EDBC server should attempt to restart a protocol driver after a protocol driver failure. The valid values are 0-10. Default is 0.
II_PROTOCOL_SNA_ LU0	The valid values are YES and NO. If the logical symbol is defined as YES, it specifies that the installation is using the SNA LU0 protocol for communications between the IBM and the remote EDBC system. An example is: `II_PROTOCOL_SNA_LU0 = YES;`
II_PROTOCOL_SNA_ LU62	The valid values are YES and NO. If the logical symbol is defined as YES, it specifies that the installation is using the SNA LU62 protocol for communications between the IBM and the remote EDBC system. An example is: `II_PROTOCOL_SNA_LU62 = YES;`
II_PROTOCOL_TCP_ IBM	The valid values are YES and NO. If the logical symbol is defined as YES, it specifies that the installation is using the IBM TCP/IP protocol for communications between the IBM and the remote EDBC system. An example is: `II_PROTOCOL_TCP_IBM = YES;`
II_PROTOCOL_TCP_ IBM_BUFSIZE	The IBM TCP/IP packet size in kilobytes. Valid values are 4-32. Default is 8. An example is: II_PROTOCOL_TCP_IBM_BUFSIZE = 16;
II_PROTOCOL_TCP_ KNET	The valid values are YES and NO. If the logical symbol is defined as YES, it specifies that the installation is using the KNET TCP/IP protocol for communications between the IBM and the remote EDBC system. An example is: `II_PROTOCOL_TCP_KNET = YES;`
II_PROTOCOL_TCP_ SNS	The valid values are YES and NO. YES specifies that the installation is using the Interlink SNS/TCP protocol for communications between OS/390 and the remote EDBC system. An example is: `II_PROTOCOL_TCP_SNS = YES;`
II_PROTOCOL_TCP_ SNS_BUFSIZE	The SNS TCP/IP packet size in kilobytes. Valid values are 4-32. Default is 8. An example is: II_PROTOCOL_TCP_SNS_BUFSIZE = 16;

II_PSF_POOL

This specifies the number of 4 KB pages that should be allocated to the Parser Facility (PSF). This parameter should be set to 30 to support Microsoft Access and Visual Basic clients. If this value is set too low, clients receive message “E_RD0001 Not enough memory for RDF temporary work area”. The default value is 25 4 KB pages. An example of the default entry is:

```
II_PSF_POOL = 25;
```

II_QEP_SIZE

This specifies the maximum memory used by the Query Execution Facility (QEF) to build data segment headers. The Query Execution Facility manages and executes query plans of non-relational gateway products, and contains the information required by QEF to execute the query. The default is 3072 bytes. An example is:

```
II_QEP_SIZE = 3072;
```

II_RECALL

This logical symbol specifies whether the gateway server should retrieve any archived data sets, through an exit, prior to completing its initialization. The sample JCL to compile and link IIRECALL exit is in EDBC.V2R3.SAMPLE.CNTL (ASMRCALL).

The valid values are YES and NO. If the logical symbol is defined as YES, it invokes the IIRECALL exit.

Note: The IIRECALL exit must be compiled and linked into EDBC.V2R3.BACK.LOAD using sample JCL.

II_SECURITY

This specifies the type of OS/390 host security used by the gateway server. The values that can be specified are CA-ACF2, RACF, CA-TSS, or NONE. An example is:

```
II_SECURITY = RACF;
```

If no value or an invalid value is specified, then the default is NONE.

II_SMFID

This specifies the value of the SMF record number that will be generated during the writing of user accounting records. This value should always be set to 0. If this value is 0, it disables the writing of SMF records. If a value is given that falls between 128 and 255 inclusive, then an SMF accounting record is written using this record ID.

Note: This function has not been enabled for the current release of the gateway products. This information is provided for planning purposes. An example is:

```
II_SMFID = 0;
```

II_TIMEZONE

This specifies the difference in hours between Greenwich Mean Time (GMT) and the local time zone where the gateway server is running. If this logical is not specified, it defaults to 0. An example is:

```
II_TIMEZONE = 7;
```

II_UTEXE_DEF

This specifies the location of front-end initialization parameters. This file must be allocated and available during execution of gateway user interfaces on OS/390. It is distributed as an empty file. It is recommended that the symbol be entered as follows:

```
II_UTEXE_DEF = "'EDBC.V2R3.FILES.UTEXE.DEF'";
```

ING_MODE

This specifies the execution mode for gateway programs. There are two values that are normally used. These are:

- SERVER

  This specifies that the gateway initializes in server mode. It is used in the following circumstances:

  – EDBC initialization

    Used when the EDBC address space is initializing.

  – Local front-end access from TSO or Batch

    Used when a front end wishes to locally access the gateway server address space. Needed by utilities such as iinamu and netu to access the gateway name server.

- NOSERVER

  This specifies that the gateway initializes in no-server mode. It is used in the following circumstances:

  – CREATPR processing.

    This mode is set in the SABExxxx member of the logical symbol library for formatting and initializing database areas and partitions.

  – TSO or Batch access to the non-relational gateway products.

    This mode is set if single-user access to the non-relational gateway products is desired.

The following shows the value that is specified in the ISVRxxxx member for this symbol.

```
ING_MODE = SERVER;
```

LOCK_LISTS

This specifies an internal non-relational gateway parameter. It must be entered as follows:

```
LOCK_LISTS = 2047;
```

LOCK_MAX

This specifies an internal non-relational gateway parameter. It must be entered as follows:

```
LOCK_MAX = 32768;
```

LOCK_RESTAB	This specifies an internal non-relational gateway parameter. It must be entered as follows: `LOCK_RESTAB = 2047;`
LOCK_TABSIZE	This specifies an internal non-relational gateway parameter. It must be entered as follows: `LOCK_TABSIZE = 2047;`
LOG_CPINTVL	This specifies an internal non-relational gateway parameter. It must be entered as follows: `LOCK_CPINTVL = 4;`
LOG_SBLKS	This specifies an internal non-relational gateway parameter. It must be entered as follows: `LOCK_SBLKS = 48;`
MAX_LOCKS	This specifies an internal non-relational gateway parameter. It must be entered as follows: `MAX_LOCKS = 350;`
SRV_MAXSERVERS	This specifies the number of gateway utility sub-tasks that will be created to perform functions required by the gateway server. This number should be set equal to at least the number of 370 central processors on OS/390. The default value is: `SRV_MAXSERVERS = 4;`
SRV_MAXTHREADS	This specifies the maximum number of concurrent threads that the gateway address space can support. The system reserves the first 16 threads for its own use. Each local connection requires a single thread. Each remote connection requires a single thread. Thus, specify the maximum number of expected connections plus 16 to arrive at this value. The default value of 64 will allow 48 concurrent user connections to be established. The default value is: `SRV_MAXTHREADS = 64;`
SRV_STKSIZE	This specifies an internal gateway parameter. Specify this value exactly as follows: `SRV_STKSIZE = 48;`
SRV_TMOUTINT	This specifies the interval in seconds that the timeout server will check for timed-out lock requests or hung protocol driver sessions. Values between 1 and 10 are valid; the default is 10 seconds. `SRV_TMOUTINT = 10;`

SYS_INGRES	This specifies the gateway data set prefix for installation files. The default value is: `SYS_INGRES = 'EDBC.V2R3';`

SABExxxx or Unnnnnnn Logical Symbol Member

This member is created for each unique database area that is to be formatted and accessed. This member is concatenated after the ISVREDBC (or equivalent) member and specifies access to a specific area. You must use this member when running the createpr utility. It is also required when accessing the non-relational gateway from TSO or as a batch job.

II_DBMS_SERVER	This parameter is required when accessing the non-relational gateway in single-user mode. It contains the OS/390 subsystem name. An example is: `II_DBMS_SERVER = EDBC;`
ING_MODE	This parameter specifies that the gateway is to run in single user mode. It must be entered exactly as follows: `ING_MODE = NOSERVER;`

Optional Logical Symbols

There are a number of logical symbols that are not normally used during the operation of the gateway. It is possible to specify these logicals to modify the behavior of the gateway. This section describes some of these logicals.

II_GCCI1_LOGLVL

This symbol specifies the level of logging to be performed. The value that is specified determines what is placed in the public log. The public log is identified by the logical symbol II_GCCI1_LOG.

The values that can be specified are 0, 4 and 6. Their meanings are:

- 0: Silent (log nothing)
- 4: Log the following:
  - GCC START/STOP status messages
  - Fatal GCC errors that cause the GCC process to stop
  - Connection-specific errors (that break a specific connection).
- 6: Same as 4, but also log connection setup and termination messages for the application layer and the transport layer.

  This value will default to the equivalent specification. An example is:

```
    II_GCCI1_LOGLVL = 4;
```

II_GCCI1_ERRLVL	This symbol specifies the level of error logging to be performed. The value given is a threshold. If a value of 0 is given, then no error logging will be done. If a value of 4 is given, then error messages with assigned levels of 0 through 4 will be logged.The default value is 4 (log all messages). An example is: `II_GCCI1_ERRLVL = 4;`
II_GW_ERRLVL	This symbol specifies the level of error logging to be performed. The value given is a threshold. If a value of 0 is given, then error with assigned level 0 will be logged. If a value of 3 is given, then error messages with assigned levels of 0 through 3 will be logged. The default value is 4 (log all messages). An example is: `II_GW_ERRLVL = 4;`

IIPARM Clist Description

The IIPARM Clist works in conjunction with the logical symbol library to configure the correct logical symbols to access the gateway. This Clist is used by the IGWFDBA utilities. This Clist allocates the following files, which are necessary for the execution of the gateway under TSO:

- IIPARM: The gateway server logical symbol file, which consists of one or more concatenated data sets
- IIERLOG: EDBC server messages
- IIVDBLOG: verifydb message log
- IIGWTR: Gateway messages
- PRINTQRY: Contains all SQL queries when the command set printqry is issued

Description of IIPARM Parameters

The permissible parameters are used to dynamically configure the allocation sequence of the IIPARM file. The acceptable parameters are as follows:

Parameter	**Description**
ISVR( )	Sets default server access symbol member. If not null, it will be appended to ISVR and concatenated to IIPARM allocation.
DBNM( )	Sets database location symbols. If not null, it will be appended to DBNM and concatenated to IIPARM allocation.
ERLOG( )	Specifies the IIERLOG allocation status. Can have one of three values: ■ ERLOG (DUMMY)—Specifies that IIERLOG will be allocated as a dummy data set. This is the default value. ■ ERLOG (*)—Specifies that IIERLOG will be allocated to the TSO terminal.
ERLOG(dsname)	Specifies that IIERLOG will be allocated to a predefined data set. If name is not in quotes, then TSO will prefix the name with the value of the TSO user prefix.
VDBLOG( )	Specifies the IIVDBLOG allocation status. Can have one of three values: ■ VDBLOG (dummy)—Specifies that IIVDBLOG be allocated as a dummy data set. This is the default value. ■ VDBLOG (*)—Specifies that IIVDBLOG be allocated to the TSO terminal. ■ VDBLOG (dsname)—Specifies that IIVDBLOG be allocated to a predefined data set. If dsname is not in quotes, then TSO will prefix the name with the value of the TSO user prefix.
SABE	Sets logical symbols to access a specific gateway database area and sort work data set. The symbol name has this form: SABEnnnn

Parameter	Description
USER	Sets logical symbols to access a specific gateway database area and sort work data set. The symbol name has this form: Unnnnnnn
FREE	Frees all previously allocated files by filename.
HELP	Shows this output.
DEBUG	Shows internal execution of IIPARM Clist.
INSRV( )	Sets the gateway server access symbol member. This parameter is similar to the ISVR( ). Unlike ISVR (), this parameter allows an explicit member that contains the server initialization parameters to be specified. It is primarily used by the IGWFDBA utility.
AREA( )	Sets the gateway area symbol member. This parameter is similar to the USER () and SABE () parameters. However, unlike the other parameters, this parameter allows the full member name to be specified. It is primarily used by the IGWFDBA utility.
SILENT	Suppresses messages displayed when the IIPARM Clist executes. When this parameter is specified, no notification messages are displayed. It is primarily used by the IGWFDBA utility.
STOGRP( )	Specifies that the Storage Group Catalog data set is to be allocated as the IISTOGRP ddname. The allowed values are: ■ YES—Allocates the IISTOGRP and IISTOLOG catalogs. If this value is specified, then the IISTOGRP ddname is allocated using the default data set name EDBC.V2R3.FILES.IISTOGRP and the IISTOLOG ddname is allocated using the default data set name EDBC.V2R3.FILES.IISTOLOG. ■ NO—Allocates the IISTOGRP and IISTOLOG catalogs as dummy data sets.

IIPARM Customization

The IIPARM Clist is customized during the IIVP stage2 jobstream execution. The ISVR (), PREFIX (), and STOGRP () parameters have defaults. The resulting Clist is placed in EDBC.V2R3.FILES.CLIST.

This data set is a fixed-block data set with an LRECL of 80. You should be able to access the IIPARM member with the SYSPROC ddname. If so, the defaults are consistent with the installation-specific naming conventions.

Examples of IIPARM Use

After you have copied this Clist into the production Clist libraries and customized it, you can use it to allocate the logical symbol file. The HELP option describes its functions. The following provides two examples of how this Clist is used. This discussion assumes that the logical symbol file has been configured and the Clist is being accessed from TSO.

Example 1

Set up the environment to access the gateway Comm server using the following command:

```
%iiparm isvr(edbc)
```

This allocates the IIPARM file to provide access to the EDBC server that was started using the parameters in the ISVREDBC member in the logical symbol file.

The following messages will appear on the TSO console:

```
**IIPARM** IIPARM DSN = 'EDBC.V2R3.FILES.IIPARM'
**IIPARM** IIPARM MBR = 'ISVREDBC'
**IIPARM** ALL FILES ALLOCATED SUCCESSFULLY
```

Example 2

Set up the environment to access the database areas for EDCUSR1. This step assumes that member UEDCUSR1 in the logical symbol library has been customized.

```
%iiparm isvr(edbc) user(edcusr1)
```

This allocates all files to allow access to the VSAM gateway from TSO. This user will use the database area that will be allocated by the gateway exclusively, for the duration of the connection.

The following messages appear on the TSO console:

```
**IIPARM** IIPARM DSN = 'EDBC.V2R3.FILES.IIPARM'
**IIPARM** IIPARM MBR = 'ISVREDBC'
**IIPARM** IIPARM MBR = 'UEDCUSR1'
**IIPARM** ALL FILES ALLOCATED SUCCESSFULLY
```

Appendix B

Customization

During stage1 of the installation, you provide site-specific information. As described in the installation procedure, this information is used to configure the IIVP stage2 jobstream. In addition, the site-specific information is used to customize several JCL members that are used to customize the EDBC server, thereby reducing the editing required before a sample JCL member can be used.

During the IIVP stage2 phase, the following datasets are customized with this site-specific information:

- EDBC.V2R3.SAMPLE.CNTL
- EDBC.V2R3.FILES.PROCLIB
- EDBC.V2R3.FILES.CLIST

This JCL can be submitted without any changes, but it is advisable to review it before execution.

The tables below list the JCL and its functions. The IINAMEI1, STARTI1, and EDBCI1 JCL are listed as they would be named with the default installation code, I1.

EDBC.V2R3.SAMPLE.CNTL

The SAMPLE.CNTL partitioned dataset contains JCL that can be used to customize various components of the EDBC server:

JCL Title	Function	Description
ASMACF2	CA-ACF2 exit	Customizes the CA-ACF2 security exit, IIACF. Compiles the source member IIACF2 in the dataset with the default name: EDBC.V2R3.FILES.ASM This JCL is required only if the CA-ACF2 security exit requires modification or if the IIACF load module is not present in the EDBC.V2R3.BACK.LOAD library.

JCL Title	Function	Description
ASMPSERV	IIPSERV module	Re-links the IIPSERV load module that contains the protocol server initialization parameters for all installed protocol servers. You do not need to execute this JCL after the IIVP stage2 has executed successfully. Use this JCL to make changes, if required.
ASMRACF	RACF exit	Customizes the RACF security exit, IIRACF. Compiles the source member IIRACF in the dataset with the default name: EDBC.V2R3.FILES.ASM This JCL is required only if the RACF security exit is modified.
ASMRCALL	Data Archive Retrieval exit	Customizes the Archive Retrieval exit. Compiles and link edits the source member IIRECALL in the dataset with the default name:
ASMRMODE	Sample VTAM SNA logmode table	Contains VTAM MODEENT statements defining the VTAM mode tables if SNA_LU0 or SNA_LU62 protocols have been selected.
ASMTSS	CA-TSS exit	Customizes the CA-TSS security exit, IITSS. Compiles the source member IITSS in the dataset with the default name: EDBC.V2R3.FILES.ASM This JCL is required only if the CA-TSS security exit is modified.
IGWFCOPY	Copy IGWF modules from BACK.LOAD to LINKLIST	JCL to copy IGWF subsystem modules into a LINKLIST library.
IGWFXSSD	Initializes the IGWF subsystem	JCL to dynamically initialize the IGWF subsystem.
IGWFZAP	Refresh and/or map IGWF subsystem modules	JCL to dynamically refresh and/or map IGWF subsystem modules.
IIARCHIV	HSM sample recall exit	Tests the IIRECALL exit without bringing up a EDBC server. This job invokes the IIRECALL exit, produces a report, and then shuts down.
IICLEAN	Recover from a RC=20 abend of the EDBC server	This JCL should be submitted when the EDBC server terminates with a RC=20 during server installation.
IIGTFSQL	Read GTF trace	Invokes the IIGTFSQL utility.

JCL Title	Function	Description
IINAMEI1	Name server file allocation JCL	Can be used to explicitly allocate the name server files needed by the EDBC server. The files that this jobstream allocates are listed below with their default names: EDBC.V2R3.FILES.NAME.IICOMSRV.I1 EDBC.V2R3.FILES.NAME.IIDB2.I1 EDBC.V2R3.FILES.NAME.IIIMS.I1 EDBC.V2R3.FILES.NAME.IIINGRES.I1 EDBC.V2R3.FILES.NAME.IILOGIN.I1 EDBC.V2R3.FILES.NAME.IINODE.I1 EDBC.V2R3.FILES.NAME.IIVSAM.I1 If you execute this JCL, you must recreate any definitions entered in the netu utility. The iiname utility does not need to be run.
EDBCEDBC	Server batch JCL	Starts the server as a batch job using in-stream JCL.
RTIAPPL	Sample VTAM SNA APPL definition	Contains the VTAM APPL statements defining the VTAM application node names if SNA_LU0 or SNA_LU62 protocols are selected.
STARTI1	Gateway server batch JCL	Allows the EDBC server to be submitted as a batch job. Assumes that the EDBCI1 member in the gateway PROCLIB has been copied to an installation user PROCLIB.
UPD01PSB catalog	Update the IIGW01PSB catalog	JCL to update the VSAM partition processing options.

EDBC.V2R3.FILES.CLIST

The EDBC.V2R3.FILES.CLIST partitioned dataset contains customized procedures required by the EDBC server:

JCL Title	Function	Description
IIPARM	Set up access to gateway from TSO	Customizes the EDBC.V2R3.FILES.CLIST Clist by the IIVP stage2 process to provide the correct defaults for the following parameters: ■ ISVR—This parameter is customized with the correct value for the OS/390 subsystem value specified in the IIVP stage1 by the IGWFINET SUBSYS= parameter. ■ PREFIX—This parameter is customized with the correct value for the OS/390 dataset prefix value specified in the IIVP stage1 by the IGWFBLD PREFIX= parameter. This Clist should be used to allocate the logical symbols needed to access the EDBC server from a batch or TSO job. The Clist library is assumed to be FB/80.

EDBC.V2R3.FILES.PROCLIB

The EDBC.V2R3.FILES.PROCLIB dataset contains procedures that are used by OS/390 TSO or batch users of the EDBC server:

JCL title	Function	Description
EDBCI1	Customization EDBC server cataloged procedure	Used to start the EDBC server as a started task or batch job. Should be copied to an installation user proclib.

Appendix C

Multiple Central Version Support

When the CA-IDMS cross memory interface is utilized, a single EDBC server can interface to multiple CA-IDMS Central Versions and/or multiple segments within a Central Version.

This interface is described in detail in the introduction in the "Working with CA-IDMS-Data" chapter. A single EDBC server can support a maximum of 50 CVs and/or segments. All the CVs must reside on the same LPAR. CVs can be at different release levels (minimum release level 12.1).

Central Version Number

The Central Version Number is the key component in the design and implementation of multiple CV/segment support. It is used in combination with the dbname to direct queries to the proper CV. It is also used to set runtime logicals that enable the CA-IDMS gateway to operate properly in different CV environments.

Installation Procedure

The installation procedure for the EDBC server for IDMS establishes an interface to a single *primary* CV. Support for additional *secondary* CV(s) is activated by performing the following post-installation tasks:

1. Install the EDBC CA-IDMS gateway system catalogs into the target CV/segment.

   If the catalogs are already installed, go to Step 2.

   a. Copy and rename the customized stage1 input (IGWFSTGS) used for the install of the EDBC server/IDMS gateway.

b. Edit the renamed IGWFSTGS and make the following changes:

   - IGWFJOB

     Change the JOBNAME= parameter so that existing members of STAGE2.CNTL are not replaced by the stage1 assembly performed in the next step.

   - IGWFUSER

     Add/delete IGWFUSER entries as appropriate.

   - IGWFIDMS

     Set INSTALL = NEW and modify the CV-related parameters as required.

c. Perform a stage1 assembly using the renamed IGWFSTGS.

d. Run stage2 jobs S0 thru S2.

2. Add and define the cross-memory modules to the secondary CV being activated.

   a. Run stage2 job S4 created in Step 1 or the S4 job used for the install of the primary CV interface (make a copy and change the SYSLMOD and IDMSLIB libraries).

   b. Perform an IDMS SYSGEN, incorporating the definitions created by the S4 job.

3. Add the following logical symbols to the ISVR member of IIPARM:

   Required Logicals

   IDMS_CV_DBNAME_x = 'yyyyyyyy';

   or

   IDMS_CV_DBNAME_x = 'CVnnn_yyyyyyyy';

   where:

x	= Relative dbname number (from 1 to a maximum of 50)
nnn	= Central Version CV number
yyyyyyyy	= 1 to 8 character segment name

   Example: IDMS_CV_DBNAME_2 = 'CV183_PRODDB1';

   Note: CVnnn need only be specified if the target CV number is different from that of the default CV number (see below).

IDMS_DEFAULT_CVNUM = 'nnn';

where:
nnn = CV number to direct a connect request if the CV number is not specified as part of the dbname

Example: IDMS_DEFAULT_CVNUM = '182';

Optional Logicals

```
dbname_CT_OPTION          =
dbname_CI_OPTION          =
dbname_SYSTEM_OWNER       =
dbname_SECSGON            =
```

where:
dbname = the value specified in the IDMS_CV_dbname_x logical

Example: CV183_PRODDB1_SYSTEM_OWNER = '$EDBC';

Note: These logicals only need to be added if the values differ from the values of the "primary" CV (IDMS). For example, if the SYSTEM_OWNER for CV183_PRODDB1 is $EDBC and IDMS_SYSTEM_OWNER = '$EDBC', then the logical CV183_PRODDB1_SYSTEM_OWNER does not need to be specified.

4. Recycle the CV and the EDBC server.
5. Connect to the activated "secondary" CV from a remote client.

Appendix

D Installing Multiple Gateways

The EDBC installation procedure supports the installing of multiple gateways in a single stage1/stage2 run. To install multiple gateways perform the following:

1. Download the files from the product tape as outlined in "Installing the Gateway" chapter.
2. Edit member IGWFSTGA in library EDBC.V2R3.FILES.ASM and make the following changes:
    a. The stage1 input statements for all the gateways are included in IGWFSTGA. Delete or comment out the statements for the gateways which are **not** being installed.
    b. Customize the stage1 input for the remaining gateway entries (see the Customizing the Stage1 Input section of the "Installing the Gateway" chapter of the appropriate *EDBC Installation and Operations Guide*).
    c. Modify the PRODUCTS= operand of the IGWFBLD statement. Delete the product codes for gateways that are **not** being installed.
3. Run stage1.

   Customize and submit member IGWFSTGA of library EDBC.V2R3.SAMPLE.CNTL.
4. Run stage2.

   Submit jobs in library EDBC.V2R3.SAMPLE.CNTL (see the Submitting the Stage2 Jobstream Jobs section of the "Installing the Gateway" chapter of the appropriate *EDBC Installation and Operations Guide*).
5. Perform the final installation and verification procedures as described in the "Installing the Gateway" chapter of the appropriate *EDBC Installation and Operations Guide*.

Index